

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 $\{t\}$. 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 $\{n\}$ that has the same orientation as $\{t\}$ 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 $\{b\}$ 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:

$$m \left({}^b \dot{v}_{P/t}^b + \omega_{b/t}^b \times v_{P/t}^b \right) = F_b^b \quad (1)$$

where:

- m is the multicopter's mass.
- $v_{P/t}^b = [u \ v \ w]^T$ is the velocity of P wrt $\{t\}$ in $\{b\}$.
- ${}^b \dot{v}_{P/t}^b = [\dot{u} \ \dot{v} \ \dot{w}]^T$ is the acceleration of P wrt $\{t\}$ in $\{b\}$.
- $\omega_{b/t}^b = [p \ q \ r]^T$ is the angular velocity of $\{b\}$ wrt $\{t\}$ in $\{b\}$.
- $F_b^b = [F_{b,x}^b \ F_{b,y}^b \ F_{b,z}^b]^T$ is the force applied on P in $\{b\}$.

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

$$v_{P/t}^n = \left(\mathbf{R}_n^b \right)^T v_{P/t}^b = [\dot{x} \ \dot{y} \ \dot{z}]^T \quad (2)$$

where:

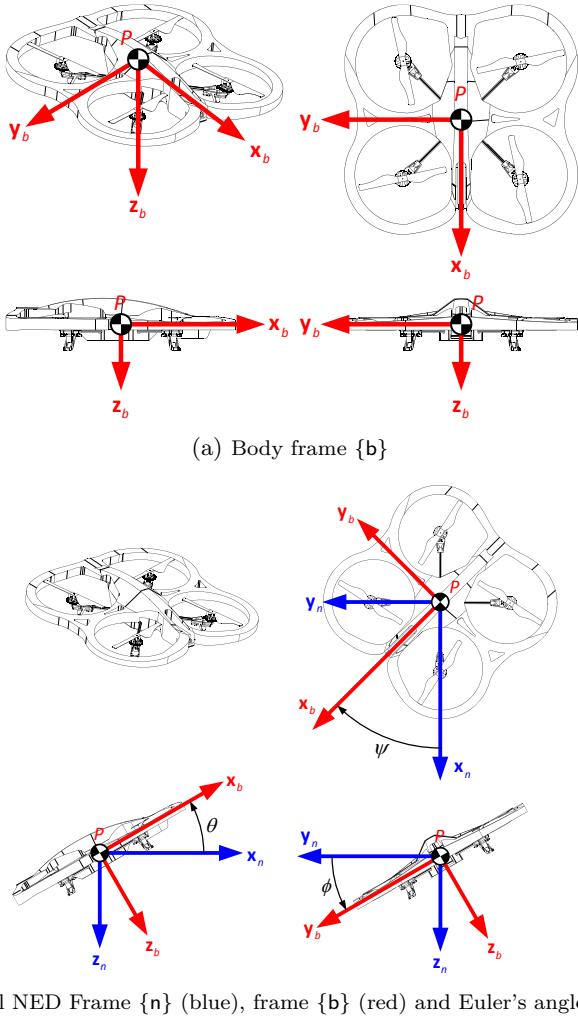


Figure 2: Body frame, local NED frame and tangent NED frame

- $\mathbf{R}_n^b = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ -c_\phi s_\psi + s_\phi s_\theta c_\psi & c_\phi c_\psi + s_\phi s_\theta s_\psi & s_\phi c_\theta \\ s_\phi s_\psi + c_\phi s_\theta c_\psi & -s_\phi c_\psi + c_\phi s_\theta s_\psi & c_\phi c_\theta \end{bmatrix}$ 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:

$$\mathbf{I}^b \ddot{\omega}_{b/t}^b + \omega_{b/t}^b \times (\mathbf{I} \omega_{b/t}^b) = \mathbf{M}_b^b \quad (3a)$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi / \cos \theta & \cos \phi / \cos \theta \end{bmatrix} \omega_{b/t}^b \quad (3b)$$

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 .
- ${}^b \dot{\omega}_{b/t}^b = [\dot{p} \ \dot{q} \ \dot{r}]^\top$ 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.
- $\mathbf{M}_b^b = [L \ M \ N]^\top$ 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 $i = 1 \dots n_{\text{mot}}$):

$$\frac{di_a}{dt} = L_a^{-1} (\mathbf{V}_a - k_e \boldsymbol{\omega} - R_a i_a) \quad (4a)$$

$$\dot{\boldsymbol{\omega}} = J_m^{-1} (k_m i_a - B_m \boldsymbol{\omega} - k_D \boldsymbol{\omega}^2) \quad (4b)$$

when $L_a \neq 0$ and:

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

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, $\mathbf{i}_a = [i_{a,1} \dots i_{a,i} \dots i_{a,n_{\text{mot}}}]^\top$. 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).
- $\boldsymbol{\omega}$ 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/rad²/s².
- J_m is the rotor and propeller's inertia in kg.m².
- 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]:

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

where:

- $\mathbf{g}^n = [0 \ 0 \ g]^\top$ is the gravity vector in $\{n\}$.
- k_T is the propeller's thrust coefficient in N/rad²/s².
- n_{mot} is the number of motors.
- ω_i is the angular speed of the i^{th} propeller.

In eq. (6), the term $m \mathbf{R}_n^b \mathbf{g}^n$ 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 $\{b\}$ is given by [10, pp.49,51-54]:

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

with:

$$\boldsymbol{\tau}_i^b = [0 \ 0 \ (-1)^{i-1} \tau_i]^\top \quad \tau_i = k_D \omega_i^2 + J_m \dot{\omega}_i \quad (7b)$$

$$\mathbf{G}_i^b = \boldsymbol{\sigma}_i^b \times \boldsymbol{\omega}_{b/t}^b \quad \boldsymbol{\sigma}_i^b = [0 \ 0 \ (-1)^i \omega_i]^\top \quad (7c)$$

where \mathbf{PP}_i^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.

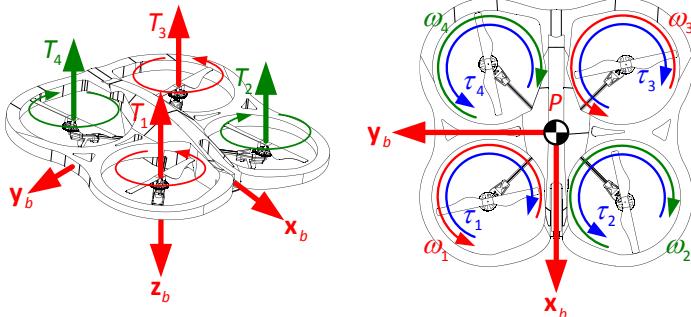


Figure 3: Thrust $T_i = k_T \omega_i^2$ and induced moments $\boldsymbol{\tau}_i$

E. Simulink Implementation

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

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

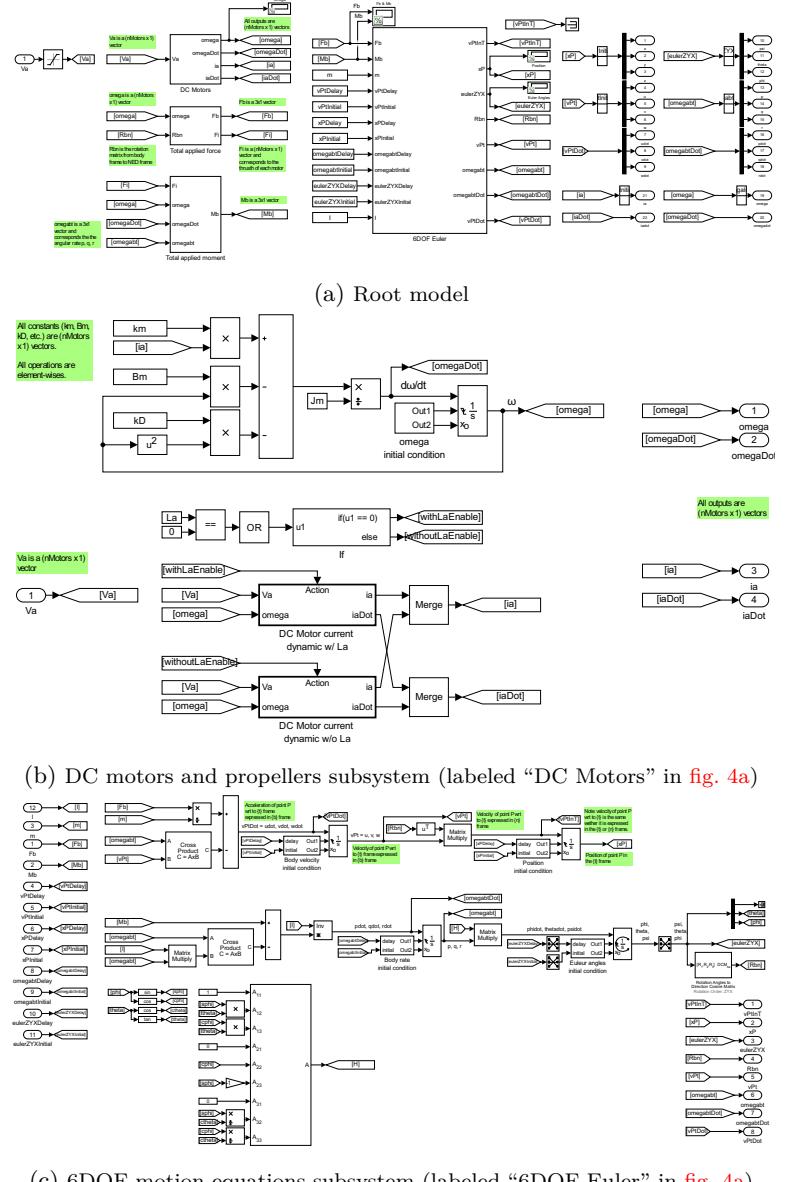


Figure 4: Simulink implementation of the nonlinear dynamics

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.

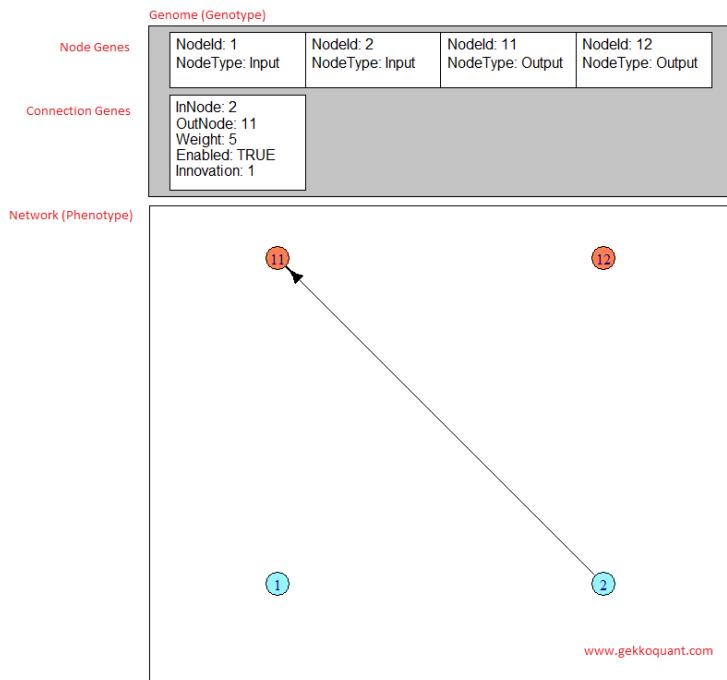


Figure 5: NEAT genome composition with node and connection genes (image source: [2])

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.

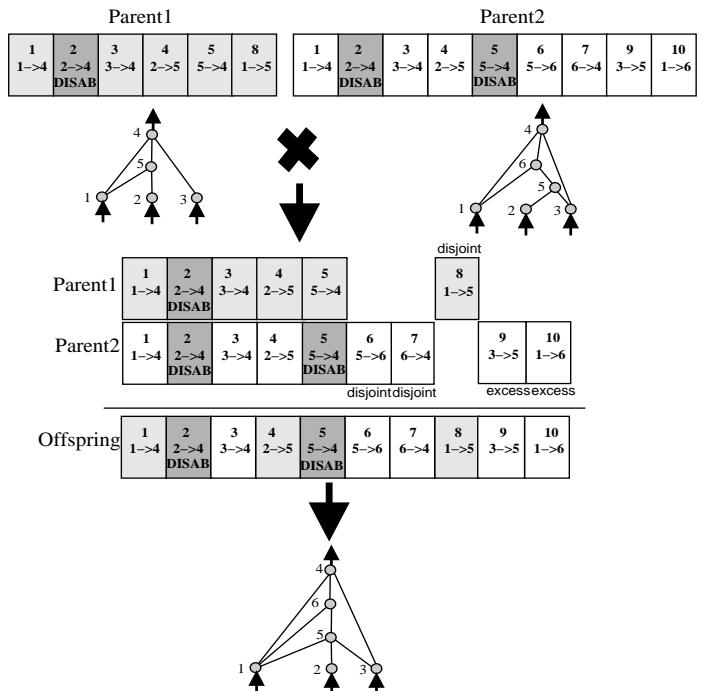


Figure 6: Matching up genomes for different network topologies using innovation numbers (image source: [12], p.109)

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:

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

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 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.

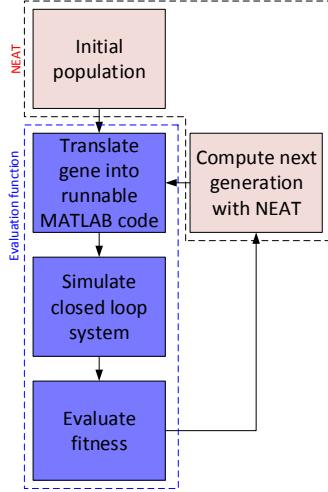


Figure 7: Algorithm overview

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.

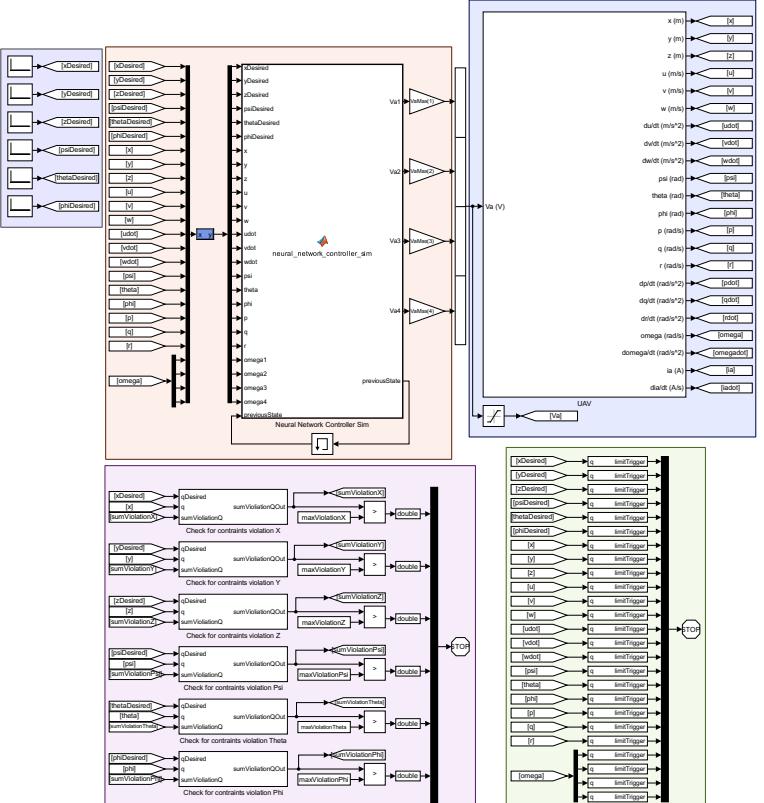


Figure 8: Simulink diagram used for ANN controller closed loop simulations

Each simulation consists of 7 trajectories¹:

- 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 $(x, y, \psi, \theta, \phi) = (50m, 50m, 0, 0, 0)$.
- Maintain an altitude of 0m for 15s with $(x, y, \psi, \theta, \phi) = (-50m, -50m, 0, 0, 0)$.
- Maintain an altitude of 0m for 15s with $(x, y, \psi, \theta, \phi) = (0, 0, 80^\circ, 0, 0)$.
- Maintain an altitude of 0m for 15s with $(x, y, \psi, \theta, \phi) = (0, 0, -80^\circ, 0, 0)$.

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 $\pm 3^\circ$ for ψ , then the constraints violation in Simulink would instead be using $\pm 8^\circ$. 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 40th generation

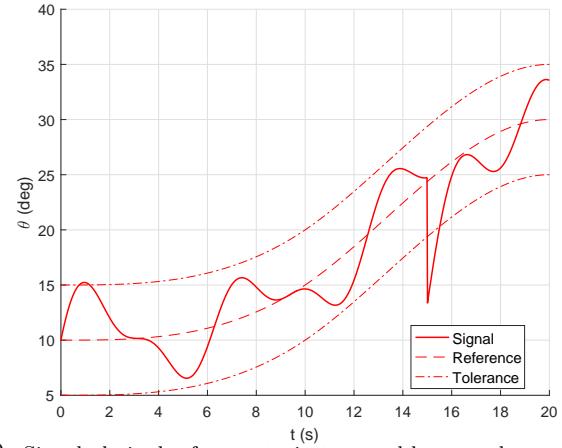


Figure 9: Signal, desired reference trajectory and lower and upper tolerances

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.

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 15th 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 30th 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 75th, 90th and 115th 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 (1st row, 2nd column) and average number of disabled connections (2nd row, 3rd 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 (3rd row), we see that the number of species suddenly jumps around the 5th 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 5th generation, there have been enough mutations to categorize them in species. Finally, we observe that the average number of hidden nodes (2nd row, 1st column), connections (2nd row, 3rd column) and recurrent connections (3rd row, 1st column) increase almost monotonically. This is expected behavior with NEAT.

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

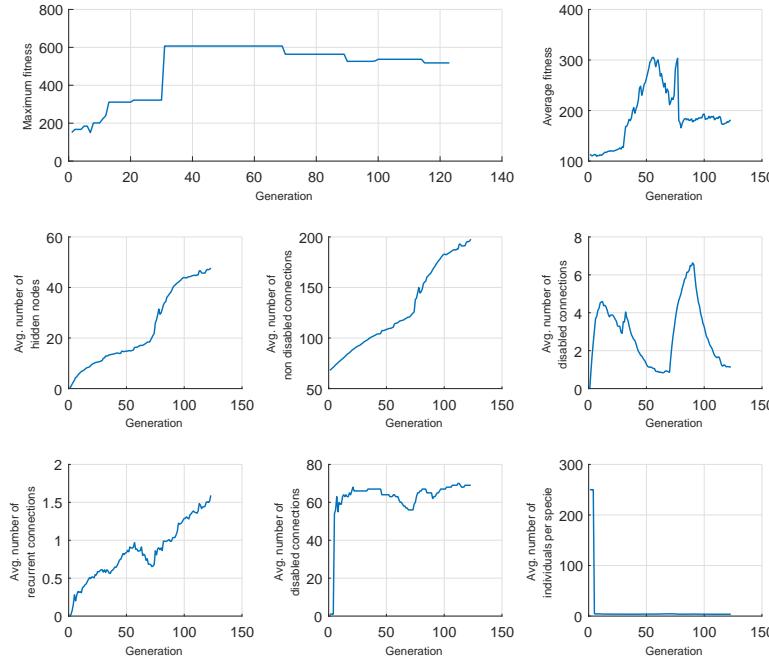
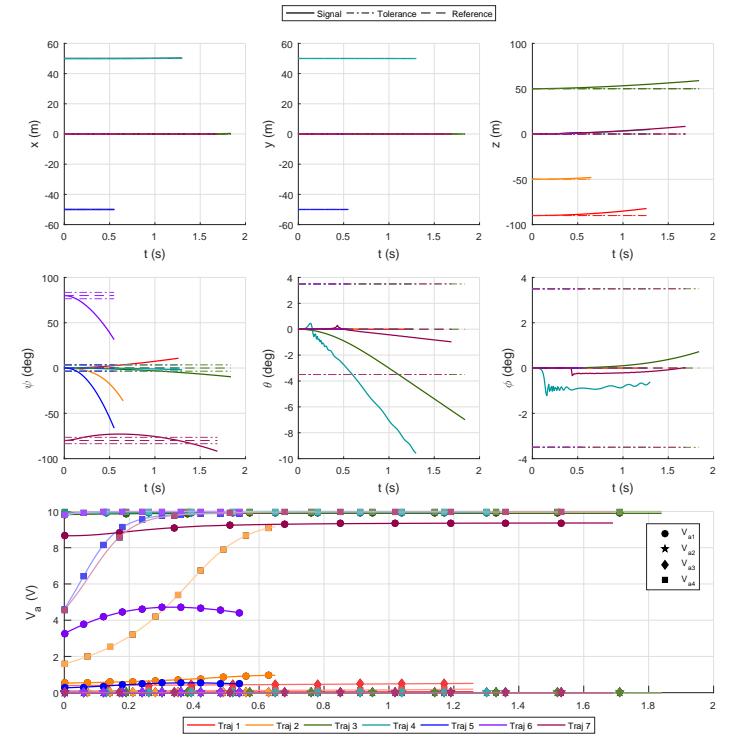


Figure 10: Evolution progress

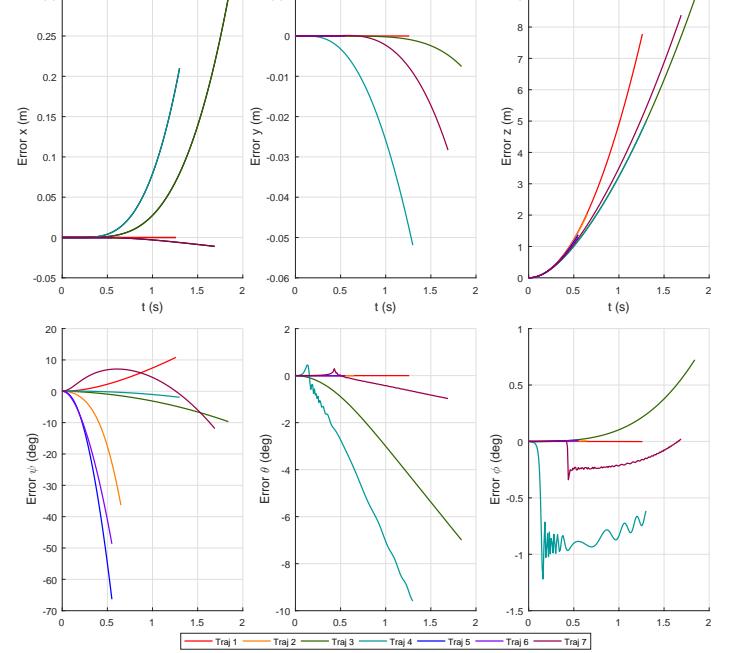
Figure 11 on page 7 presents the best individual's performance at the 1st generation. At this stage, there are no hidden nodes and all the weights are initialized randomly. We observe that the resulting controller performs very badly and for a very short amount of time before the simulation is interrupted 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 ascent². 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

²Note 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.

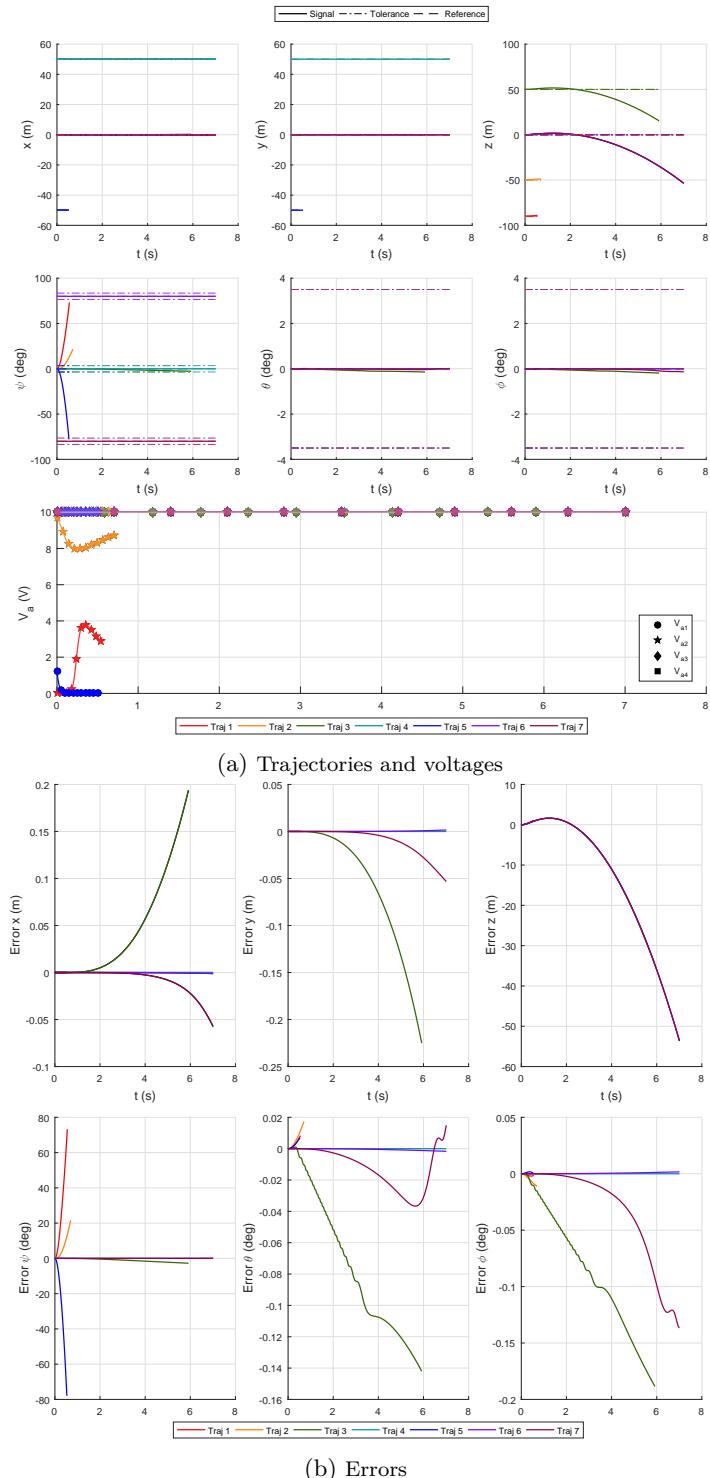


Figure 12: Best individual's simulation results at generation 9

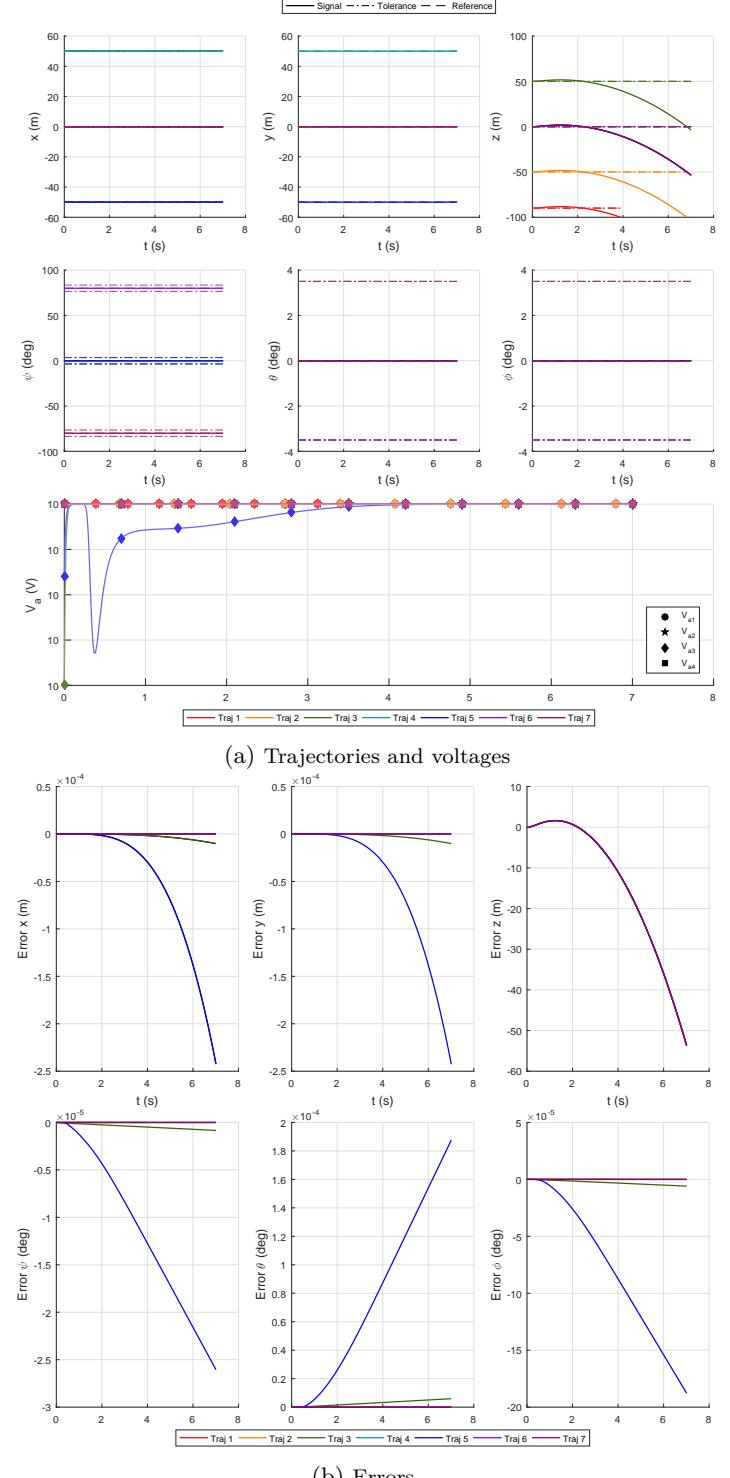


Figure 13: Best individual's simulation results at generation 49

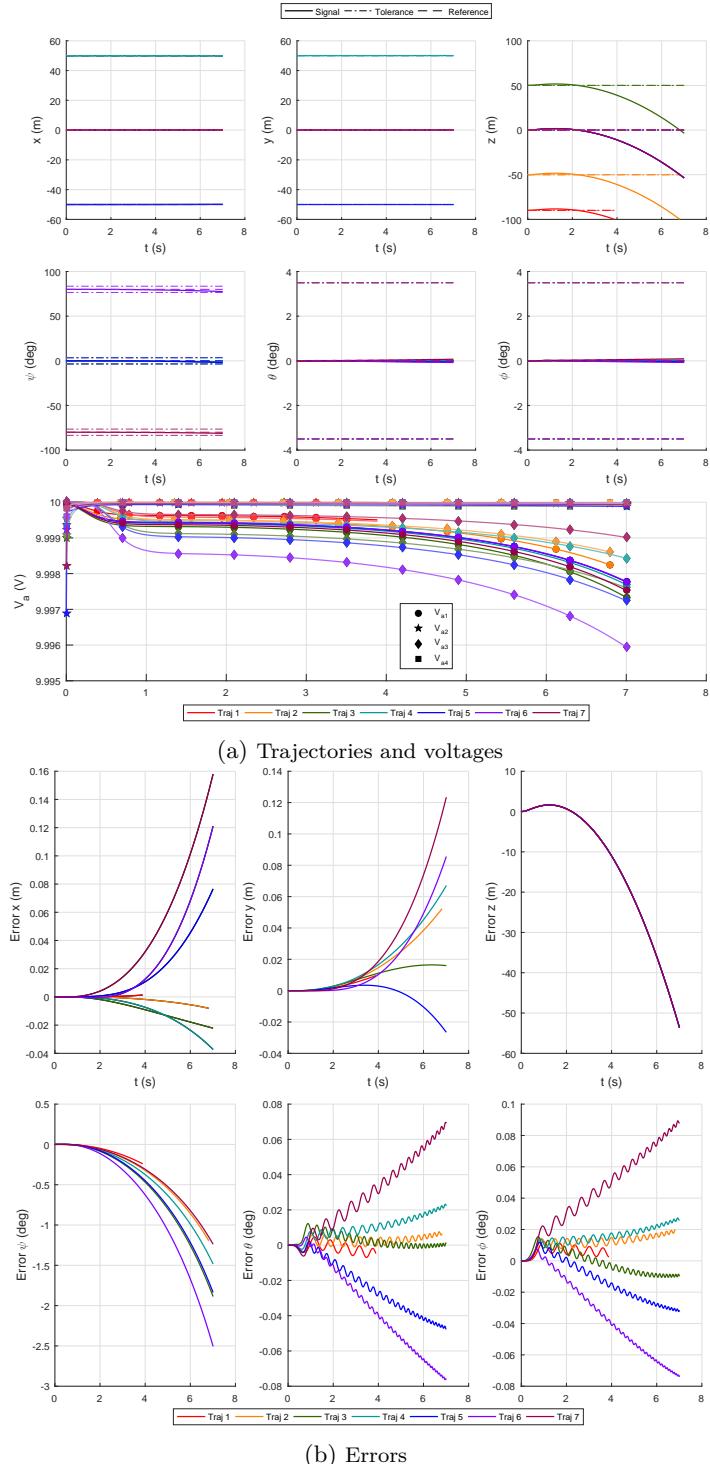


Figure 14: Best individual's simulation results at generation 91

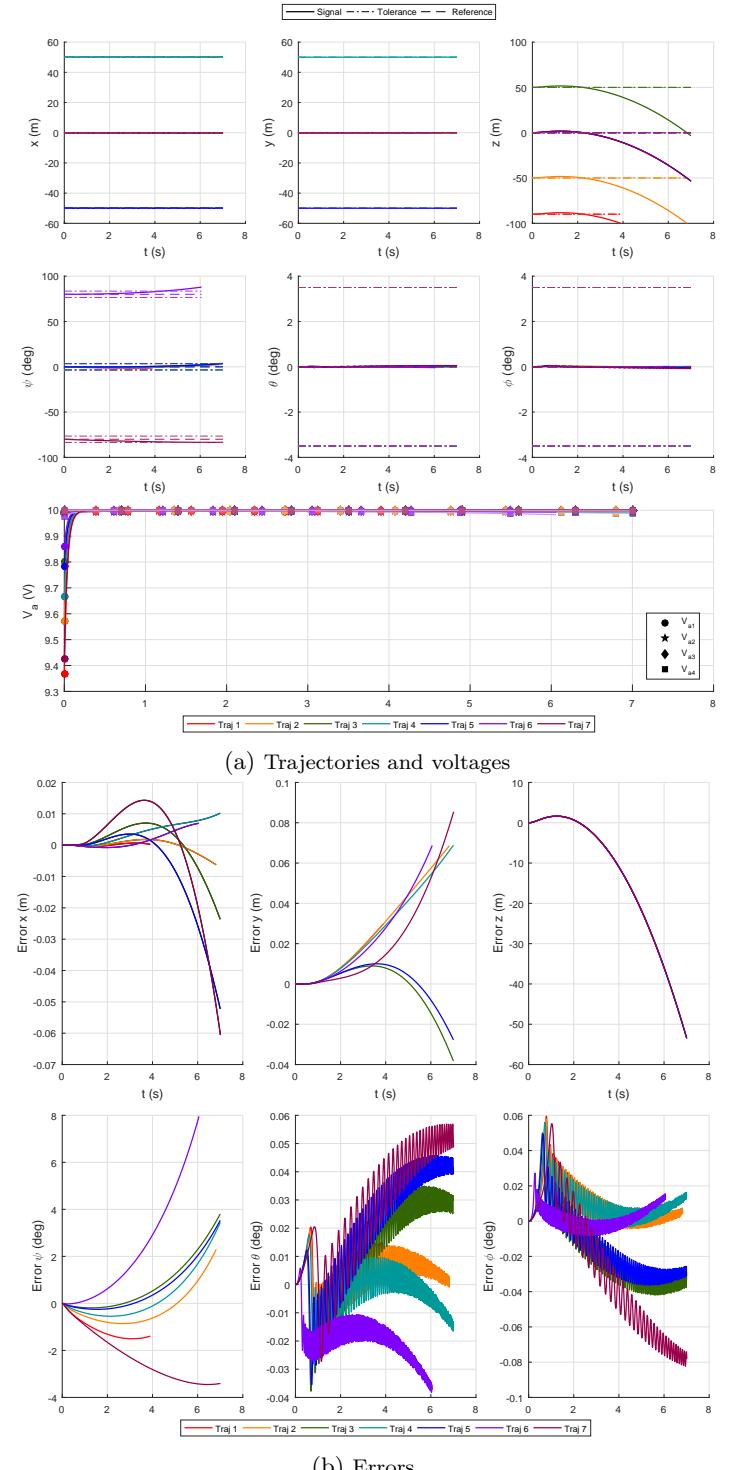
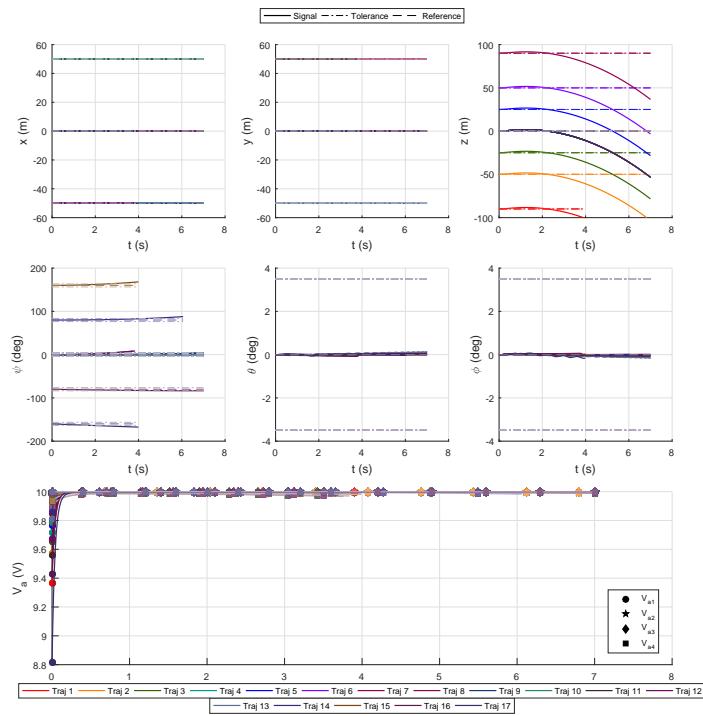
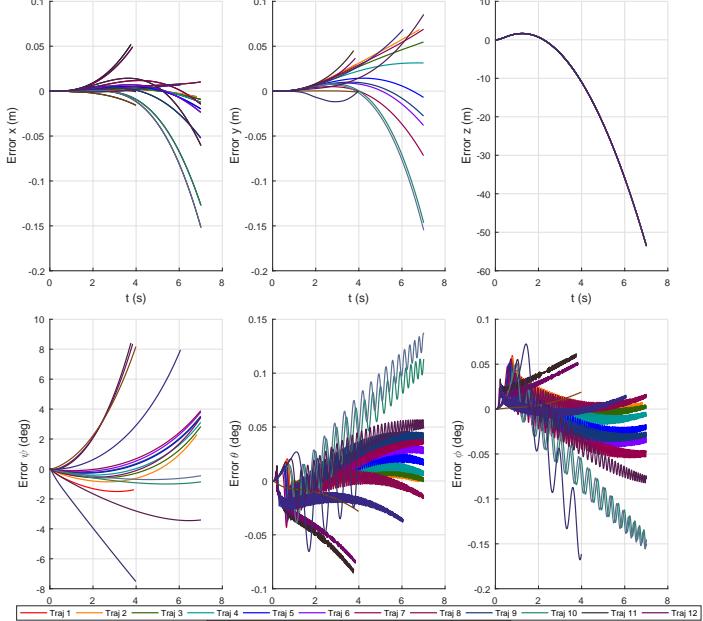


Figure 15: Best individual's simulation results at generation 123



(a) Trajectories and voltages



(b) Errors

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

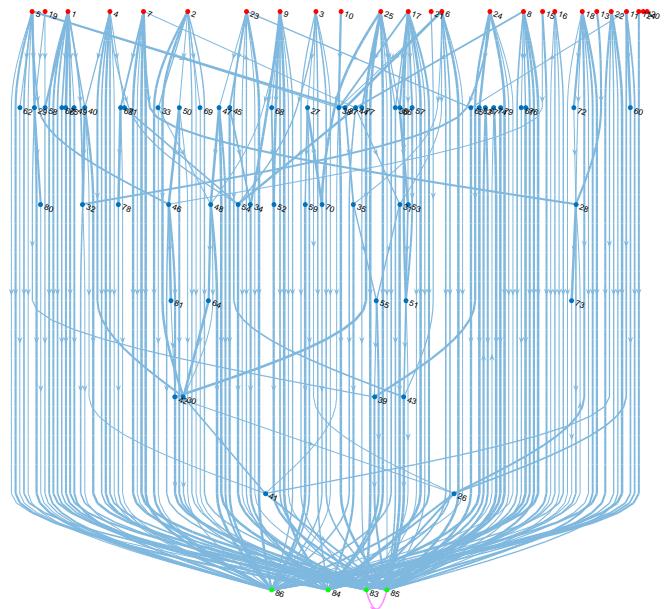


Figure 17: Best neural network topology at last generation

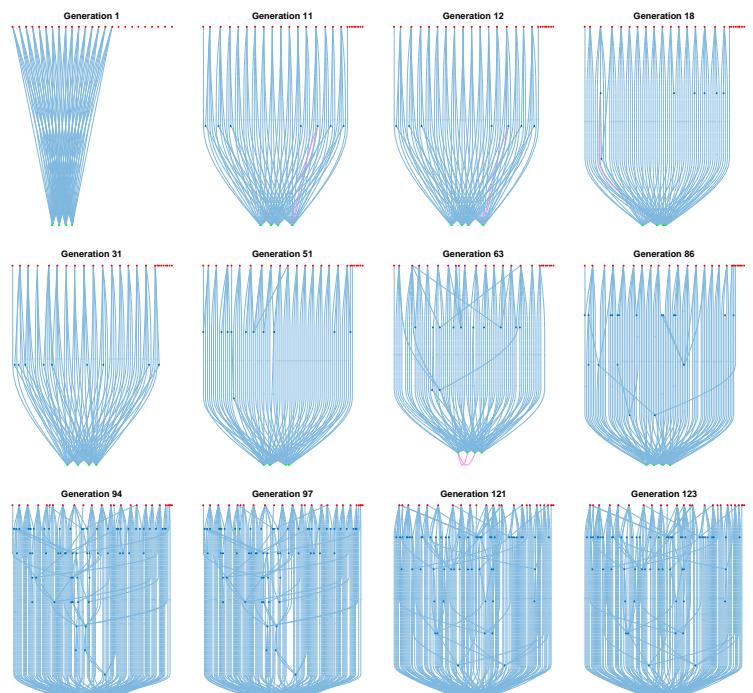


Figure 18: Topology evolution for 12 of the best neural networks throughout generations

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 174**. The first row (**line 175**) is the species' ID, the second (**line 176**) is the allotted number of offspring for the species and the last row (**line 177**) indicates the actual number of offspring.

```

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 (AGL1FMBM16JJG74ENNFF69M8E99B1A)
76 nPreviousStates = 1 for individual 132 in generation 2 (8C341J2LHOBL7A31HONFMG3JA779P)
77 nPreviousStates = 1 for individual 103 in generation 2 (D9634597G34E1B6BNFBLE5E820HNNA)
78 nPreviousStates = 1 for individual 92 in generation 2 (68GOIGB160N8NK9DFDCFABNNMOLF)
79 nPreviousStates = 1 for individual 174 in generation 2 (EK59N1IBJM49KLLHPJ09C8K3KGJ7F5)
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 (699BJJ32F576GLPSBG5I99A8N9CP)
105 nPreviousStates = 1 for individual 64 in generation 3 (CND1EMG544482M2991I0M219D5F7H0)
106 .....
107 nPreviousStates = 1 for individual 236 in generation 3 (JK9DC5JK3H5A5F501CMLFEL19EONG)
108 nPreviousStates = 1 for individual 225 in generation 3 (7ALJOJ9B096L00A2FH03079NNGFJ7K)
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 (L636H10EC048577FBJDL61E0995NF0)
134 nPreviousStates = 1 for individual 101 in generation 4 (16PNDLKNL6OBEP6F08EB6MLIE33926)
135 .....
136 nPreviousStates = 1 for individual 224 in generation 4 (A957BDI2G91PP2INL26DKE980BIBN)
137 nPreviousStates = 1 for individual 231 in generation 4 (B16H313C24BB8C57GF0300EHAJEODE)
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

```

```

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

```

```

159 Mutation.probabilityRecurrency = 0.05000
160 Mutation.probabilityMutateWeight = 0.90000
161
162 nPreviousStates = 1 for individual 91 in generation 5 (CA9DPK32L0D0C0E4GL138CB84A10LH)
163 nPreviousStates = 1 for individual 88 in generation 5 (87DIEAIGAE5G99K8F92PPFKESHG722)
164 .....
165 nPreviousStates = 1 for individual 239 in generation 5 (4HD2GI3F10DPJ8EOH208CC24LIN2FM)
166 nPreviousStates = 1 for individual 238 in generation 5 (C69J6IJ8F0B16C05P56A3L35G769NK)
167 fitness = 107.42 for individual 52 in generation 5
168 fitness = 112.34 for individual 39 in generation 5
169 .....
170 fitness = 156.03 for individual 239 in generation 5
171 fitness = 119.25 for individual 66 in generation 5
172 Max fitness is 184.580714
173
174 Existing and propagating matrix:
175 1 2 3 4 5 6 7 8 9 10 11 .... 43 44 45 46 47 48 49 50 51 52 53 54
176 4 5 5 4 5 5 5 5 4 .... 4 5 5 4 5 4 5 4 5 4 5 4 5
177 1 1 1 1 1 1 1 1 0 0 .... 0 0 0 0 0 0 0 0 0 0 0 0
178
179 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 process (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

```

1 clear all
2 close all
3 clc
4
5 % Uncomment for repeatability between runs
6 % rng(6541651)
7 rng('shuffle');
8
9 % Constants
10 DBG_2_RAD = pi/180;
11 RAD_2_DBG = 180/pi;
12 MM_2_M = 1/1000;
13 M_2_MM = 1000/1;
14
15 % Date and time
dateAndTime = datestr(datetime('now','TimeZone','local',...
    'Format','d-MMM-y HH:mm:ss Z'));
dateAndTime = strrep(dateAndTime,'_','_');
dateAndTime = strrep(dateAndTime,'_','_');
21
22 % Set up diary
23 if ~exist('diary','dir')
24     mkdir('diary');
25 end
diary(['diary','/design_controller','_',dateAndTime,'.txt']);
27 fprintf('Diary stored in %s\n',...
28     ['diary','/design_controller','_',dateAndTime,'.txt']);
29
30 % Close all figure handles
31 figAllHandle = findall(0,'Type','figure');
32 if ~isempty(figAllHandle)
33     for iFigHandle = 1:numel(figAllHandle)
34         % figAllHandle(iFigHandle).CloseRequestFcn = @closereq;
35         set(0,'currentFigure', figAllHandle(iFigHandle));
36         close('force');
37         % close(figAllHandle(iFigHandle));
38         delete(figAllHandle(iFigHandle));
39     end
40     clear figAllHandle;
41 end
42
43 %% Delete paths from MATLAB search path
44
45 % Code snippet from https://goo.gl/0QwRmN
46
47 % Get a list of all files and folders in this folder.
48 if exist('temp_uav_nn_controller','dir')
49     files = dir('temp_uav_nn_controller');
50     files(1:2) = [];
51 end
52
53 % Get a logical vector that tells which is a directory.
54 if exist('temp_uav_nn_controller','dir')
55     dirFlags = [files.isdir];
56 end
57
58 % Extract only those that are directories.
59 if exist('temp_uav_nn_controller','dir')
60     subfolders = files(dirFlags);

```

```

61     end
62
63 % Remove path
64 if exist('temp_uav_nn_controller','dir')
65     for k = 1:length(subFolders)
66         rmpath(['temp_uav_nn_controller',subFolders(k).name]);
67     end
68 end
69 rehash();
70
71 %% Parameters - General
72
73 % If set to 1, will load population, generation, innovation_record and
74 % species_record from neatSave.mat at start of algorithm, if set to 0,
75 % species_record will start with initial population, new species record and new
76 % innovation record, at generation = 1 (default option),
77 loadFlag = 1;
78
79 % If set to 1, will save population, generation, innovation_record and
80 % species_record to neatSave.mat at every generation (default option)
81 saveFlag = 1;
82
83 % If set to 1, NEAT will start with the gene from geneFile and initialize the
84 % weight randomly between [-1,+1]
85 loadGene = 0;
86 geneFile = 'neat_data/neatSave_27_Nov_2016_20_27_44_043.mat';
87
88 % Maximum number of generations for generational loop
89 nMaxGeneration = 200; % 200
90
91 % Population size
92 populationSize = 250; % 150, 500, 300
93
94 % Input and output nodes numbers
95 nInputNodes = 25;
96 nOutputNodes = 4;
97
98 % Vector of initially connected input nodes out of complete number of input nodes
99 % (if you want to start with a subset and let evolution decide which ones are necessary)
100 % for a fully connected initial population, list all the inputs. For each input
101 % node listed in initialConnectedInputNodes, it will be connected to every
102 % output node during population's initialization.
103 % initialConnectedInputNodes = 1:nInputNodes;
104 % initialConnectedInputNodes = [1,2,3,4,5,6,7,8,9,16,17,18,22,23,24,25];
105 % initialConnectedInputNodes = 1;
106
107 % Fitness limit
108 fitnessLimit = Inf;
109
110 %% Parameters - Speciation
111
112 % Speciation parameters as published by Ken Stanley
113 Speciation.c1 = 1.0;
114 Speciation.c2 = 1.0;
115 Speciation.c3 = 0.4; % 0.4
116 Speciation.threshold = 25.0; % 3 (10.0,12.0,14.0)
117
118 %% Parameters - Reproduction - Stagnation + Refocus
119
120 % Threshold to judge if a species is in stagnation (max fitness of species
121 % varies below threshold) this threshold is of course dependent on your
122 % fitness function, if you have a fitness function which has a large
123 % spread, you might want to increase this threshold
124 Stagnation.threshold = 1e-2;
125
126 % If max fitness of species has stayed within Stagnation.threshold in the
127 % last Stagnation.nGenerations generations, all its fitnesses will be
128 % reduced to 0, so it will die out
129 Stagnation.nGenerations = 15; % 15
130
131 % Computation is done the following way: the absolute difference between
132 % the average max fitness of the last Stagnation.nGenerations
133 % generations and the max fitness of each of these generations is computed
134 % and compared to Stagnation.threshold.
135 % If it stays within this threshold for the indicated number of
136 % generations, the species is eliminated
137 Refocus.threshold = 1e-2;
138
139 % If maximum overall fitness of population doesn't change within threshold
140 % for this number of generations, only the top two species are allowed to
141 % reproduce
142 Refocus.nGenerations = 20; % 20
143
144 %% Parameters - Reproduction - Initial
145
146 % The percentage of each species which will be eliminated (lowest
147 % performing individuals). This percentage for eliminating individuals will
148 % only be used in species which have more individuals than numberForKill
149 Initial.killPercentage = 0.2;
150
151 % Please note that whatever the above settings, the code always ensures
152 % that at least 2 individuals are kept to be able to cross over, or at
153 % least the single individual in a species with only one individual
154 Initial.numberForKill = 5;
155
156 % Species which have equal or greater than numberCopy individuals will
157 % have their best individual copied unchanged into the next generation
158 Initial.numberCopy = 5;
159
160 %% Parameters - Reproduction - Selection
161
162 % Selection (ranking and stochastic universal sampling)
163 % Number between 1.1 and 2.0, determines selective pressure towards most
164 % fit individual of species
165 Selection.pressure = 2;
166
167 %% Parameters - Reproduction - Crossover
168
169 % Percentage governs the way in which new population will be composed from
170 % old population. exception: species with just one individual can only use
171 % mutation
172 Crossover.percentage = 0.8;
173
174 % If crossover has been selected, this probability governs the
175 % intra/interspecies parent composition being used for the
176 % standard-crossover in which matching connection genes are inherited
177 % randomly from both parents. In the (1-Crossover.probabilityMultipoint)
178 % cases, weights of the connection genes are the mean of the
179 % corresponding parent genes
180 Crossover.probabilityInterspecies = 0.001;
181 Crossover.probabilityMultipoint = 0.6;
182
183 %% Parameters - Reproduction - Mutation
184
185 % Adding node probability
186 Mutation.probabilityAddNode = 0.60; % 0.03
187 Mutation.probabilityAddNodeMin = 0.05; % 0.03
188 Mutation.probabilityAddNodeShrinkRate = ...
189     (Mutation.probabilityAddNode-Mutation.probabilityAddNodeMin)/60;
190 if true
191     Mutation.probabilityAddNodeScheduling = [...
192         0005, 10, 15, 20, 30, 40, 70, 75, 90, 100, Inf;
193         0.90, 0.50, 0.40, 0.30, 0.20, 0.10, 0.90, 0.85, 0.80, 0.10, 0.05];
194 else
195     Mutation.probabilityAddNodeScheduling = -1;
196 end
197
198 % Adding connection probability
199 Mutation.probabilityAddConnection = 0.80; % 0.05
200 Mutation.probabilityAddConnectionMin = 0.20;
201 Mutation.probabilityAddConnectionShrinkRate = ...
202     (Mutation.probabilityAddConnection-Mutation.probabilityAddConnectionMin)/60;

```

```

203 if true
204   Mutation.probabilityAddConnectionScheduling = [...,
205     0.005, 10, 15, 20, 30, 40, 70, 75, 90, Inf;
206     0.90, 0.70, 0.60, 0.45, 0.40, 0.30, 0.90, 0.85, 0.80, 0.20];
207 else
208   Mutation.probabilityAddNodeScheduling = -1;
209 end
210
211 % If we are in add_connection_mutation, this governs if a recurrent
212 % connection is allowed. Note: this will only activate if the random
213 % connection is a recurrent one, otherwise the connection is simply
214 % accepted. If no possible non-recurrent connections exist for the
215 % current node genes, then for e.g. a probability of 0.1, 9 times out
216 % of 10 no connection is added.
217 Mutation.probabilityRecurrency      = 0.05; % 0.05
218 Mutation.probabilityRecurrencyMin    = Mutation.probabilityRecurrency;
219 Mutation.probabilityRecurrencyShrinkRate = ...;
220   (Mutation.probabilityRecurrency-Mutation.probabilityRecurrencyMin)/60;
221 Mutation.probabilityMutateWeight     = 0.9; % 0.9
222 Mutation.probabilityMutateWeightMin   = 0.9;
223 Mutation.probabilityMutateWeightShrinkRate = ...;
224   (Mutation.probabilityMutateWeight-Mutation.probabilityMutateWeightMin)/60;
225
226 % Weights will be restricted from -Mutation.weightCap to Mutation.weightCap
227 Mutation.weightCap = 0.5; % 8
228
229 % Random distribution with width Mutation.weightRange, centered on 0.
230 % mutation range of 5 will give random distribution from -2.5 to 2.5
231 Mutation.weightRange = 0.25; % 5
232
233 % Probability of a connection gene being reenabled in offspring if it was
234 % inherited disabled
235 Mutation.probabilityGeneReenabled = 0.10; % 0.20
236
237 %X Display parameters
238
239 if Mutation.probabilityAddNodeScheduling(1) == -1
240   mutationAddNode1Str = 'Mutation.probabilityAddNode'           = %4.5f\n';
241   mutationAddNode2Str = 'Mutation.probabilityAddNodeShrinkRate' = %4.5f\n';
242   mutationAddNode3Str = 'Mutation.probabilityAddNodeMin'        = %4.5f\n';
243   mutationAddNode1Val = {Mutation.probabilityAddNode};
244   mutationAddNode2Val = Mutation.probabilityAddNodeShrinkRate;
245   mutationAddNode3Val = Mutation.probabilityAddNodeMin;
246 else
247   mutationAddNode1Str = [...,
248     'Mutation.probabilityAddNode'           = [%s]\n',...
249     'Mutation.probabilityAddNodeShrinkRate' = [%s]\n'];
250
251   mutationAddNode2Str = 'Mutation.probabilityAddNodeShrinkRate' = %s\n';
252   mutationAddNode3Str = 'Mutation.probabilityAddNodeMin'        = %s\n';
253   mutationAddNode1Val = {Mutation.probabilityAddNode};
254   num2str(Mutation.probabilityAddNodeScheduling(1,:),'%t\t%2.2f', ...
255   num2str(Mutation.probabilityAddNodeScheduling(2,:),'%t\t%2.2f');
256   mutationAddNode2Val = 'N/A';
257   mutationAddNode3Val = 'N/A';
258 end
259
260 if Mutation.probabilityAddConnectionScheduling(1) == -1
261   mutationAddConn1 = 'Mutation.probabilityAddConnection'          = %4.5f\n';
262   mutationAddConn2 = 'Mutation.probabilityAddConnectionShrinkRate' = %4.5f\n';
263   mutationAddConn3 = 'Mutation.probabilityAddConnectionMin'        = %4.5f\n';
264   mutationAddConn1Val = {Mutation.probabilityAddConnection};
265   mutationAddConn2Val = Mutation.probabilityAddConnectionShrinkRate;
266   mutationAddConn3Val = Mutation.probabilityAddConnectionMin;
267 else
268   mutationAddConn1Str = [...,
269     'Mutation.probabilityAddConnection'          = [%s]\n',...
270     'Mutation.probabilityAddConnectionShrinkRate' = [%s]\n'];
271   mutationAddConn2Str = 'Mutation.probabilityAddConnectionMin'        = %s\n';
272   mutationAddConn1Val = {Mutation.probabilityAddConnection};
273   num2str(Mutation.probabilityAddConnectionScheduling(1,:),'%t\t%2.2f', ...
274   num2str(Mutation.probabilityAddConnectionScheduling(2,:),'%t\t%2.2f');
275   mutationAddConn2Val = 'N/A';
276   mutationAddConn3Val = 'N/A';
277 end
278
279 fprintf([...
280   'Using the following parameters:\n',...
281   'loadFlag'                                = %4.f\n',...
282   'saveFlag'                                 = %4.f\n',...
283   'nMaxGeneration'                          = %4.f\n',...
284   'fitnessLimitSize'                        = %4.f\n',...
285   'nInputNodes'                             = %4.f\n',...
286   'nOutputNodes'                           = %4.f\n',...
287   'initialConnectedInputNodes'              = [%s]\n',...
288   'fitnessLimit'                            = %4.5f\n',...
289   'Speciation.c1'                           = %4.5f\n',...
290   'Speciation.c2'                           = %4.5f\n',...
291   'Speciation.c3'                           = %4.5f\n',...
292   'Speciation.threshold'                    = %4.5f\n',...
293   'Stagnation.threshold'                   = %4.5f\n',...
294   'Stagnation.nGenerations'                = %4.f\n',...
295   'Refocus.threshold'                      = %4.5f\n',...
296   'Refocus.ngenerations'                  = %4.f\n',...
297   'Initial.killPercentage'                 = %4.5f\n',...
298   'Initial.numberForKill'                  = %4.5f\n',...
299   'Initial.numberCopy'                     = %4.5f\n',...
300   'Selection.pressure'                    = %4.5f\n',...
301   'Crossover.percentage'                 = %4.5f\n',...
302   'Crossover.probabilityInterspecies'    = %4.5f\n',...
303   'Crossover.probabilityMultipoint'       = %4.5f\n',...
304   mutationAddNode1Str,...;
305   mutationAddNode2Str,...;
306   mutationAddNode3Str,...;
307   mutationAddConn1Str,...;
308   mutationAddConn2Str,...;
309   mutationAddConn3Str,...;
310   'Mutation.probabilityRecurrency'        = %4.5f\n',...
311   'Mutation.probabilityRecurrencyShrinkRate' = %4.5f\n',...
312   'Mutation.probabilityRecurrencyMin'      = %4.5f\n',...
313   'Mutation.probabilityMutateWeight'       = %4.5f\n',...
314   'Mutation.probabilityMutateWeightShrinkRate' = %4.5f\n',...
315   'Mutation.probabilityMutateWeightMin'     = %4.5f\n',...
316   'Mutation.weightCap'                    = %4.5f\n',...
317   'Mutation.weightRange'                 = %4.5f\n',...
318   'Mutation.probabilityGeneReenabled'     = %4.5f\n',...
319 ],...
320 loadFlag,...,
321 saveFlag,...,
322 nMaxGeneration,...,
323 populationSize,...,
324 nInputNodes,...,
325 nOutputNodes,...,
326 num2str(initialConnectedInputNodes),...
327 fitnessLimit,...,
328 Speciation.c1,...,
329 Speciation.c2,...,
330 Speciation.c3,...,
331 Speciation.threshold,...,
332 Stagnation.threshold,...,
333 Stagnation.nGenerations,...,
334 Refocus.threshold,...,
335 Refocus.ngenerations,...,
336 Initial.killPercentage,...,
337 Initial.numberForKill,...,
338 Initial.numberCopy,...,
339 Selection.pressure,...,
340 Crossover.percentage,...,
341 Crossover.probabilityInterspecies,...,
342 Crossover.probabilityMultipoint,...,
343 mutationAddNode1Val{:},...
344 mutationAddNode2Val,...,
345   mutationAddNode3Val,...,
346   mutationAddConn1Val{:},...
347   mutationAddConn2Val,...,
348   mutationAddConn3Val,...,
349   Mutation.probabilityRecurrency,...,
350   Mutation.probabilityRecurrencyShrinkRate,...,
351   Mutation.probabilityRecurrencyMin,...,
352   Mutation.probabilityMutateWeight,...,
353   Mutation.probabilityMutateWeightShrinkRate,...,
354   Mutation.probabilityMutateWeightMin,...,
355   Mutation.weightCap,...,
356   Mutation.weightRange,...,
357   Mutation.probabilityGeneReenabled...,
358 ];
359
360 %X Start pool
361
362 clusterName = 'local';
363 nWorkers = 14;
364 create_parpool(clusterName, nWorkers, false);
365
366 %X Evaluation function parameters
367
368 % Handle to evaluation function
369 evalFunHandle = @uav_evaluation_function;
370
371 % Use parallel evaluation
372 evalFunParameters.useParallel = true;
373
374 % Default simulation parameters
375 evalFunParameters.SimParam.abcTol = 'auto'; % 'auto'
376 evalFunParameters.SimParam.relTol = 'auto'; % 'auto'
377 evalFunParameters.SimParam.stopTime = '10';
378 evalFunParameters.SimParam.solverType = 'Variable-step'; % 'Variable-step', 'Fixed-step'
379 evalFunParameters.SimParam.solver = 'VariableStepAuto'; % 'VariableStepAuto'
380 evalFunParameters.SimParam.ninStep = '0.000001'; % 'auto', '0.000001'
381 evalFunParameters.SimParam.maxStep = 'auto'; % 'auto'
382 evalFunParameters.SimParam.maxConsecutiveMinStep = '100'; % '1'
383 evalFunParameters.SimParam.fixedStep = evalFunParameters.SimParam.maxStep;
384 evalFunParameters.SimParam.initialStep = 'auto'; % 'auto'
385 evalFunParameters.SimParam.simulationMode = 'accelerator'; % 'normal', 'accelerator', 'rapid-accelerator'
386 evalFunParameters.SimParam.fastRestart = 'on';
387
388 % UAV parameters
389 evalFunParameters.UavParam.g = 9.80665;
390 evalFunParameters.UavParam.Motors = 4;
391 evalFunParameters.UavParam.Psi = 312;
392 evalFunParameters.UavParam.I = [0.0032373, 0, 0, 0, 0.0032373, 0, 0, 0, 0.0058673];
393 evalFunParameters.UavParam.PP = 0.1591*[1, 1, -1, -1, 1, -1, 1, 0, 0, 0];
394 evalFunParameters.UavParam.Rx = 0.5033*ones(4,1);
395 evalFunParameters.UavParam.Lx = 0.1e-3*ones(4,1);
396 evalFunParameters.UavParam.kx = 7.8e-3*ones(4,1);
397 evalFunParameters.UavParam.km = 3.7e-3*ones(4,1);
398 evalFunParameters.UavParam.KD = 1.140e-7*ones(4,1);
399 evalFunParameters.UavParam.Jm = 3.357e-5*ones(4,1);
400 evalFunParameters.UavParam.Bm = 1.23e-6*ones(4,1);
401 evalFunParameters.UavParam.KT = 2.980e-6*ones(4,1);
402 evalFunParameters.UavParam.VaMin = zeros(4,1);
403 evalFunParameters.UavParam.VaMax = 10*ones(4,1);
404
405 % Input limits
406 evalFunParameters.UavParam.xDesiredMin = -100;
407 evalFunParameters.UavParam.xDesiredMax = +100;
408 evalFunParameters.UavParam.yDesiredMin = -100;
409 evalFunParameters.UavParam.yDesiredMax = +100;
410 evalFunParameters.UavParam.zDesiredMin = -100;
411 evalFunParameters.UavParam.zDesiredMax = +100;
412 evalFunParameters.UavParam.psiDesiredMin = -pi;
413 evalFunParameters.UavParam.psiDesiredMax = +pi;
414 evalFunParameters.UavParam.thetaDesiredMin = -pi/2;
415 evalFunParameters.UavParam.thetaDesiredMax = +pi/2;
416 evalFunParameters.UavParam.phiDesiredMin = -pi;
417 evalFunParameters.UavParam.phiDesiredMax = +pi;
418 evalFunParameters.UavParam.XMin = -100;
419 evalFunParameters.UavParam.XMax = +100;
420 evalFunParameters.UavParam.YMin = -100;
421 evalFunParameters.UavParam.YMax = +100;
422 evalFunParameters.UavParam.ZMin = -100;
423 evalFunParameters.UavParam.ZMax = +100;
424 evalFunParameters.UavParam.uMin = -30;
425 evalFunParameters.UavParam.uMax = +30;
426 evalFunParameters.UavParam.udotMin = -60;
427 evalFunParameters.UavParam.udotMax = +60;
428 evalFunParameters.UavParam.vMin = -30;
429 evalFunParameters.UavParam.vMax = +30;
430 evalFunParameters.UavParam.wMin = -20;
431 evalFunParameters.UavParam.wMax = +20;
432 evalFunParameters.UavParam.udotMin = -60;
433 evalFunParameters.UavParam.udotMax = +60;
434 evalFunParameters.UavParam.vdotMin = -60;
435 evalFunParameters.UavParam.vdotMax = +60;
436 evalFunParameters.UavParam.wdotMin = -60;
437 evalFunParameters.UavParam.wdotMax = +60;
438 evalFunParameters.UavParam.psiMin = -pi;
439 evalFunParameters.UavParam.psiMax = +pi;
440 evalFunParameters.UavParam.thetaMin = -pi/2;
441 evalFunParameters.UavParam.thetaMax = +pi/2;
442 evalFunParameters.UavParam.phiMin = -pi;
443 evalFunParameters.UavParam.phiMax = +pi;
444 evalFunParameters.UavParam.pMax = -2.4;
445 evalFunParameters.UavParam.qMax = -2.4;
446 evalFunParameters.UavParam.rMax = -2.4;
447 evalFunParameters.UavParam.xMin = -4.72;
448 evalFunParameters.UavParam.xMax = +4.72;
449 evalFunParameters.UavParam.omega1Min = 0;
450 evalFunParameters.UavParam.omega1Max = +800;
451 evalFunParameters.UavParam.omega2Min = 0;
452 evalFunParameters.UavParam.omega2Max = +800;
453 evalFunParameters.UavParam.omega3Min = 0;
454 evalFunParameters.UavParam.omega3Max = +800;
455 evalFunParameters.UavParam.omega4Min = 0;
456 evalFunParameters.UavParam.omega4Max = +800;
457 % Load trajectories from trajectory_definition.m
458 trajectory_definition();
459
460 % Save mat file with EvalFunParameters
461 folderName = 'eval_fun_parameters';
462 if exist(folderName, 'dir')
463   mkdir(folderName);
464 end
465 save([folderName, '/eval_fun_parameters_', dateAndTime, '.mat'], 'EvalFunParameters');
466
467 %X Run NEAT
468
469 % Call NEAT
470 [Population, bestIndividual] = neat(...,
471   nMaxGeneration,...,
472   loadGene,...,
473   geneFile,...,
474   loadFlag,...,
475   saveFlag,...,
476   populationSize,...,
477   nInputNodes,...,
478   nOutputNodes,...,
479   initialConnectedInputNodes,...,
480   fitnessLimit,...,
481   Speciation,...,
482   Initial,...,
483   Stagnation,...,
484   Refocus,...,
485   Selection,...,
486   Crossover,...,
487   mutationAddNode1Val{:},...
488   mutationAddNode2Val,...,
489   mutationAddNode3Val,...,
490   mutationAddConn1Val{:},...
491   mutationAddConn2Val,...,
492   mutationAddConn3Val,...,
493   mutationAddConn1Val{:},...
494   mutationAddConn2Val,...,
495   mutationAddConn3Val,...,
496   mutationAddConn1Val{:},...
497   mutationAddConn2Val,...,
498   mutationAddConn3Val,...,
499   mutationAddConn1Val{:},...
500   mutationAddConn2Val,...,
501   mutationAddConn3Val,...,
502   mutationAddConn1Val{:},...
503   mutationAddConn2Val,...,
504   mutationAddConn3Val,...,
505   mutationAddConn1Val{:},...
506   mutationAddConn2Val,...,
507   mutationAddConn3Val,...,
508   mutationAddConn1Val{:},...
509   mutationAddConn2Val,...,
510   mutationAddConn3Val,...,
511   mutationAddConn1Val{:},...
512   mutationAddConn2Val,...,
513   mutationAddConn3Val,...,
514   mutationAddConn1Val{:},...
515   mutationAddConn2Val,...,
516   mutationAddConn3Val,...,
517   mutationAddConn1Val{:},...
518   mutationAddConn2Val,...,
519   mutationAddConn3Val,...,
520   mutationAddConn1Val{:},...
521   mutationAddConn2Val,...,
522   mutationAddConn3Val,...,
523   mutationAddConn1Val{:},...
524   mutationAddConn2Val,...,
525   mutationAddConn3Val,...,
526   mutationAddConn1Val{:},...
527   mutationAddConn2Val,...,
528   mutationAddConn3Val,...,
529   mutationAddConn1Val{:},...
530   mutationAddConn2Val,...,
531   mutationAddConn3Val,...,
532   mutationAddConn1Val{:},...
533   mutationAddConn2Val,...,
534   mutationAddConn3Val,...,
535   mutationAddConn1Val{:},...
536   mutationAddConn2Val,...,
537   mutationAddConn3Val,...,
538   mutationAddConn1Val{:},...
539   mutationAddConn2Val,...,
540   mutationAddConn3Val,...,
541   mutationAddConn1Val{:},...
542   mutationAddConn2Val,...,
543   mutationAddConn3Val,...,
544   mutationAddConn1Val{:},...
545   mutationAddConn2Val,...,
546   mutationAddConn3Val,...,
547   mutationAddConn1Val{:},...
548   mutationAddConn2Val,...,
549   mutationAddConn3Val,...,
550   mutationAddConn1Val{:},...
551   mutationAddConn2Val,...,
552   mutationAddConn3Val,...,
553   mutationAddConn1Val{:},...
554   mutationAddConn2Val,...,
555   mutationAddConn3Val,...,
556   mutationAddConn1Val{:},...
557   mutationAddConn2Val,...,
558   mutationAddConn3Val,...,
559   mutationAddConn1Val{:},...
560   mutationAddConn2Val,...,
561   mutationAddConn3Val,...,
562   mutationAddConn1Val{:},...
563   mutationAddConn2Val,...,
564   mutationAddConn3Val,...,
565   mutationAddConn1Val{:},...
566   mutationAddConn2Val,...,
567   mutationAddConn3Val,...,
568   mutationAddConn1Val{:},...
569   mutationAddConn2Val,...,
570   mutationAddConn3Val,...,
571   mutationAddConn1Val{:},...
572   mutationAddConn2Val,...,
573   mutationAddConn3Val,...,
574   mutationAddConn1Val{:},...
575   mutationAddConn2Val,...,
576   mutationAddConn3Val,...,
577   mutationAddConn1Val{:},...
578   mutationAddConn2Val,...,
579   mutationAddConn3Val,...,
580   mutationAddConn1Val{:},...
581   mutationAddConn2Val,...,
582   mutationAddConn3Val,...,
583   mutationAddConn1Val{:},...
584   mutationAddConn2Val,...,
585   mutationAddConn3Val,...,
586   mutationAddConn1Val{:},...
587   mutationAddConn2Val,...,
588   mutationAddConn3Val,...,
589   mutationAddConn1Val{:},...
590   mutationAddConn2Val,...,
591   mutationAddConn3Val,...,
592   mutationAddConn1Val{:},...
593   mutationAddConn2Val,...,
594   mutationAddConn3Val,...,
595   mutationAddConn1Val{:},...
596   mutationAddConn2Val,...,
597   mutationAddConn3Val,...,
598   mutationAddConn1Val{:},...
599   mutationAddConn2Val,...,
600   mutationAddConn3Val,...,
601   mutationAddConn1Val{:},...
602   mutationAddConn2Val,...,
603   mutationAddConn3Val,...,
604   mutationAddConn1Val{:},...
605   mutationAddConn2Val,...,
606   mutationAddConn3Val,...,
607   mutationAddConn1Val{:},...
608   mutationAddConn2Val,...,
609   mutationAddConn3Val,...,
610   mutationAddConn1Val{:},...
611   mutationAddConn2Val,...,
612   mutationAddConn3Val,...,
613   mutationAddConn1Val{:},...
614   mutationAddConn2Val,...,
615   mutationAddConn3Val,...,
616   mutationAddConn1Val{:},...
617   mutationAddConn2Val,...,
618   mutationAddConn3Val,...,
619   mutationAddConn1Val{:},...
620   mutationAddConn2Val,...,
621   mutationAddConn3Val,...,
622   mutationAddConn1Val{:},...
623   mutationAddConn2Val,...,
624   mutationAddConn3Val,...,
625   mutationAddConn1Val{:},...
626   mutationAddConn2Val,...,
627   mutationAddConn3Val,...,
628   mutationAddConn1Val{:},...
629   mutationAddConn2Val,...,
630   mutationAddConn3Val,...,
631   mutationAddConn1Val{:},...
632   mutationAddConn2Val,...,
633   mutationAddConn3Val,...,
634   mutationAddConn1Val{:},...
635   mutationAddConn2Val,...,
636   mutationAddConn3Val,...,
637   mutationAddConn1Val{:},...
638   mutationAddConn2Val,...,
639   mutationAddConn3Val,...,
640   mutationAddConn1Val{:},...
641   mutationAddConn2Val,...,
642   mutationAddConn3Val,...,
643   mutationAddConn1Val{:},...
644   mutationAddConn2Val,...,
645   mutationAddConn3Val,...,
646   mutationAddConn1Val{:},...
647   mutationAddConn2Val,...,
648   mutationAddConn3Val,...,
649   mutationAddConn1Val{:},...
650   mutationAddConn2Val,...,
651   mutationAddConn3Val,...,
652   mutationAddConn1Val{:},...
653   mutationAddConn2Val,...,
654   mutationAddConn3Val,...,
655   mutationAddConn1Val{:},...
656   mutationAddConn2Val,...,
657   mutationAddConn3Val,...,
658   mutationAddConn1Val{:},...
659   mutationAddConn2Val,...,
660   mutationAddConn3Val,...,
661   mutationAddConn1Val{:},...
662   mutationAddConn2Val,...,
663   mutationAddConn3Val,...,
664   mutationAddConn1Val{:},...
665   mutationAddConn2Val,...,
666   mutationAddConn3Val,...,
667   mutationAddConn1Val{:},...
668   mutationAddConn2Val,...,
669   mutationAddConn3Val,...,
670   mutationAddConn1Val{:},...
671   mutationAddConn2Val,...,
672   mutationAddConn3Val,...,
673   mutationAddConn1Val{:},...
674   mutationAddConn2Val,...,
675   mutationAddConn3Val,...,
676   mutationAddConn1Val{:},...
677   mutationAddConn2Val,...,
678   mutationAddConn3Val,...,
679   mutationAddConn1Val{:},...
680   mutationAddConn2Val,...,
681   mutationAddConn3Val,...,
682   mutationAddConn1Val{:},...
683   mutationAddConn2Val,...,
684   mutationAddConn3Val,...,
685   mutationAddConn1Val{:},...
686   mutationAddConn2Val,...,
687   mutationAddConn3Val,...,
688   mutationAddConn1Val{:},...
689   mutationAddConn2Val,...,
690   mutationAddConn3Val,...,
691   mutationAddConn1Val{:},...
692   mutationAddConn2Val,...,
693   mutationAddConn3Val,...,
694   mutationAddConn1Val{:},...
695   mutationAddConn2Val,...,
696   mutationAddConn3Val,...,
697   mutationAddConn1Val{:},...
698   mutationAddConn2Val,...,
699   mutationAddConn3Val,...,
700   mutationAddConn1Val{:},...
701   mutationAddConn2Val,...,
702   mutationAddConn3Val,...,
703   mutationAddConn1Val{:},...
704   mutationAddConn2Val,...,
705   mutationAddConn3Val,...,
706   mutationAddConn1Val{:},...
707   mutationAddConn2Val,...,
708   mutationAddConn3Val,...,
709   mutationAddConn1Val{:},...
710   mutationAddConn2Val,...,
711   mutationAddConn3Val,...,
712   mutationAddConn1Val{:},...
713   mutationAddConn2Val,...,
714   mutationAddConn3Val,...,
715   mutationAddConn1Val{:},...
716   mutationAddConn2Val,...,
717   mutationAddConn3Val,...,
718   mutationAddConn1Val{:},...
719   mutationAddConn2Val,...,
720   mutationAddConn3Val,...,
721   mutationAddConn1Val{:},...
722   mutationAddConn2Val,...,
723   mutationAddConn3Val,...,
724   mutationAddConn1Val{:},...
725   mutationAddConn2Val,...,
726   mutationAddConn3Val,...,
727   mutationAddConn1Val{:},...
728   mutationAddConn2Val,...,
729   mutationAddConn3Val,...,
730   mutationAddConn1Val{:},...
731   mutationAddConn2Val,...,
732   mutationAddConn3Val,...,
733   mutationAddConn1Val{:},...
734   mutationAddConn2Val,...,
735   mutationAddConn3Val,...,
736   mutationAddConn1Val{:},...
737   mutationAddConn2Val,...,
738   mutationAddConn3Val,...,
739   mutationAddConn1Val{:},...
740   mutationAddConn2Val,...,
741   mutationAddConn3Val,...,
742   mutationAddConn1Val{:},...
743   mutationAddConn2Val,...,
744   mutationAddConn3Val,...,
745   mutationAddConn1Val{:},...
746   mutationAddConn2Val,...,
747   mutationAddConn3Val,...,
748   mutationAddConn1Val{:},...
749   mutationAddConn2Val,...,
750   mutationAddConn3Val,...,
751   mutationAddConn1Val{:},...
752   mutationAddConn2Val,...,
753   mutationAddConn3Val,...,
754   mutationAddConn1Val{:},...
755   mutationAddConn2Val,...,
756   mutationAddConn3Val,...,
757   mutationAddConn1Val{:},...
758   mutationAddConn2Val,...,
759   mutationAddConn3Val,...,
760   mutationAddConn1Val{:},...
761   mutationAddConn2Val,...,
762   mutationAddConn3Val,...,
763   mutationAddConn1Val{:},...
764   mutationAddConn2Val,...,
765   mutationAddConn3Val,...,
766   mutationAddConn1Val{:},...
767   mutationAddConn2Val,...,
768   mutationAddConn3Val,...,
769   mutationAddConn1Val{:},...
770   mutationAddConn2Val,...,
771   mutationAddConn3Val,...,
772   mutationAddConn1Val{:},...
773   mutationAddConn2Val,...,
774   mutationAddConn3Val,...,
775   mutationAddConn1Val{:},...
776   mutationAddConn2Val,...,
777   mutationAddConn3Val,...,
778   mutationAddConn1Val{:},...
779   mutationAddConn2Val,...,
780   mutationAddConn3Val,...,
781   mutationAddConn1Val{:},...
782   mutationAddConn2Val,...,
783   mutationAddConn3Val,...,
784   mutationAddConn1Val{:},...
785   mutationAddConn2Val,...,
786   mutationAddConn3Val,...,
787   mutationAddConn1Val{:},...
788   mutationAddConn2Val,...,
789   mutationAddConn3Val,...,
790   mutationAddConn1Val{:},...
791   mutationAddConn2Val,...,
792   mutationAddConn3Val,...,
793   mutationAddConn1Val{:},...
794   mutationAddConn2Val,...,
795   mutationAddConn3Val,...,
796   mutationAddConn1Val{:},...
797   mutationAddConn2Val,...,
798   mutationAddConn3Val,...,
799   mutationAddConn1Val{:},...
800   mutationAddConn2Val,...,
801   mutationAddConn3Val,...,
802   mutationAddConn1Val{:},...
803   mutationAddConn2Val,...,
804   mutationAddConn3Val,...,
805   mutationAddConn1Val{:},...
806   mutationAddConn2Val,...,
807   mutationAddConn3Val,...,
808   mutationAddConn1Val{:},...
809   mutationAddConn2Val,...,
810   mutationAddConn3Val,...,
811   mutationAddConn1Val{:},...
812   mutationAddConn2Val,...,
813   mutationAddConn3Val,...,
814   mutationAddConn1Val{:},...
815   mutationAddConn2Val,...,
816   mutationAddConn3Val,...,
817   mutationAddConn1Val{:},...
818   mutationAddConn2Val,...,
819   mutationAddConn3Val,...,
820   mutationAddConn1Val{:},...
821   mutationAddConn2Val,...,
822   mutationAddConn3Val,...,
823   mutationAddConn1Val{:},...
824   mutationAddConn2Val,...,
825   mutationAddConn3Val,...,
826   mutationAddConn1Val{:},...
827   mutationAddConn2Val,...,
828   mutationAddConn3Val,...,
829   mutationAddConn1Val{:},...
830   mutationAddConn2Val,...,
831   mutationAddConn3Val,...,
832   mutationAddConn1Val{:},...
833   mutationAddConn2Val,...,
834   mutationAddConn3Val,...,
835   mutationAddConn1Val{:},...
836   mutationAddConn2Val,...,
837   mutationAddConn3Val,...,
838   mutationAddConn1Val{:},...
839   mutationAddConn2Val,...,
840   mutationAddConn3Val,...,
841   mutationAddConn1Val{:},...
842   mutationAddConn2Val,...,
843   mutationAddConn3Val,...,
844   mutationAddConn1Val{:},...
845   mutationAddConn2Val,...,
846   mutationAddConn3Val,...,
847   mutationAddConn1Val{:},...
848   mutationAddConn2Val,...,
849   mutationAddConn3Val,...,
850   mutationAddConn1Val{:},...
851   mutationAddConn2Val,...,
852   mutationAddConn3Val,...,
853   mutationAddConn1Val{:},...
854   mutationAddConn2Val,...,
855   mutationAddConn3Val,...,
856   mutationAddConn1Val{:},...
857   mutationAddConn2Val,...,
858   mutationAddConn3Val,...,
859   mutationAddConn1Val{:},...
860   mutationAddConn2Val,...,
861   mutationAddConn3Val,...,
862   mutationAddConn1Val{:},...
863   mutationAddConn2Val,...,
864   mutationAddConn3Val,...,
865   mutationAddConn1Val{:},...
866   mutationAddConn2Val,...,
867   mutationAddConn3Val,...,
868   mutationAddConn1Val{:},...
869   mutationAddConn2Val,...,
870   mutationAddConn3Val,...,
871   mutationAddConn1Val{:},...
872   mutationAddConn2Val,...,
873   mutationAddConn3Val,...,
874   mutationAddConn1Val{:},...
875   mutationAddConn2Val,...,
876   mutationAddConn3Val,...,
877   mutationAddConn1Val{:},...
878   mutationAddConn2Val,...,
879   mutationAddConn3Val,...,
880   mutationAddConn1Val{:},...
881   mutationAddConn2Val,...,
882   mutationAddConn3Val,...,
883   mutationAddConn1Val{:},...
884   mutationAddConn2Val,...,
885   mutationAddConn3Val,...,
886   mutationAddConn1Val{:},...
887   mutationAddConn2Val,...,
888   mutationAddConn3Val,...,
889   mutationAddConn1Val{:},...
890   mutationAddConn2Val,...,
891   mutationAddConn3Val,...,
892   mutationAddConn1Val{:},...
893   mutationAddConn2Val,...,
894   mutationAddConn3Val,...,
895   mutationAddConn1Val{:},...
896   mutationAddConn2Val,...,
897   mutationAddConn3Val,...,
898   mutationAddConn1Val{:},...
899   mutationAddConn2Val,...,
900   mutationAddConn3Val,...,
901   mutationAddConn1Val{:},...
902   mutationAddConn2Val,...,
903   mutationAddConn3Val,...,
904   mutationAddConn1Val{:},...
905   mutationAddConn2Val,...,
906   mutationAddConn3Val,...,
907   mutationAddConn1Val{:},...
908   mutationAddConn2Val,...,
909   mutationAddConn3Val,...,
910   mutationAddConn1Val{:},...
911   mutationAddConn2Val,...,
912   mutationAddConn3Val,...,
913   mutationAddConn1Val{:},...
914   mutationAddConn2Val,...,
915   mutationAddConn3Val,...,
916   mutationAddConn1Val{:},...
917   mutationAddConn2Val,...,
918   mutationAddConn3Val,...,
919   mutationAddConn1Val{:},...
920   mutationAddConn2Val,...,
921   mutationAddConn3Val,...,
922   mutationAddConn1Val{:},...
923   mutationAddConn2Val,...,
924   mutationAddConn3Val,...,
925   mutationAddConn1Val{:},...
926   mutationAddConn2Val,...,
927   mutationAddConn3Val,...,
928   mutationAddConn1Val{:},...
929   mutationAddConn2Val,...,
930   mutationAddConn3Val,...,
931   mutationAddConn1Val{:},...
932   mutationAddConn2Val,...,
933   mutationAddConn3Val,...,
934   mutationAddConn1Val{:},...
935   mutationAddConn2Val,...,
936   mutationAddConn3Val,...,
937   mutationAddConn1Val{:},...
938   mutationAddConn2Val,...,
939   mutationAddConn3Val,...,
940   mutationAddConn1Val{:},...
941   mutationAddConn2Val,...,
942   mutationAddConn3Val,...,
943   mutationAddConn1Val{:},...
944   mutationAddConn2Val,...,
945   mutationAddConn3Val,...,
946   mutationAddConn1Val{:},...
947   mutationAddConn2Val,...,
948   mutationAddConn3Val,...,
949   mutationAddConn1Val{:},...
950   mutationAddConn2Val,...,
951   mutationAddConn3Val,...,
952   mutationAddConn1Val{:},...
953   mutationAddConn2Val,...,
954   mutationAddConn3Val,...,
955   mutationAddConn1Val{:},...
956   mutationAddConn2Val,...,
957   mutationAddConn3Val,...,
958   mutationAddConn1Val{:},...
959   mutationAddConn2Val,...,
960   mutationAddConn3Val,...,
961   mutationAddConn1Val{:},...
962   mutationAddConn2Val,...,
963   mutationAddConn3Val,...,
964   mutationAddConn1Val{:},...
965   mutationAddConn2Val,...,
966   mutationAddConn3Val,...,
967   mutationAddConn1Val{:},...
968   mutationAddConn2Val,...,
969   mutationAddConn3Val,...,
970   mutationAddConn1Val{:},...
971   mutationAddConn2Val,...,
972   mutationAddConn3Val,...,
973   mutationAddConn1Val{:},...
974   mutationAddConn2Val,...,
975   mutationAddConn3Val,...,
976   mutationAddConn1Val{:},...
977   mutationAddConn2Val,...,
978   mutationAddConn3Val,...,
979   mutationAddConn1Val{:},...
980   mutationAddConn2Val,...,
981   mutationAddConn3Val,...,
982   mutationAddConn1Val{:},...
983   mutationAddConn2Val,...,
984   mutationAddConn3Val,...,
985   mutationAddConn1Val{:},...
986   mutationAddConn2Val,...,
987   mutationAddConn3Val,...,
988   mutationAddConn1Val{:},...
989   mutationAddConn2Val,...,
990   mutationAddConn3Val,...,
991   mutationAddConn1Val{:},...
992   mutationAddConn2Val,...,
993   mutationAddConn3Val,...,
994   mutationAddConn1Val{:},...
995   mutationAddConn2Val,...,
996   mutationAddConn3Val,...,
997   mutationAddConn1Val{:},...
998   mutationAddConn2Val,...,
999   mutationAddConn3Val,...,
1000   mutationAddConn1Val{:},...
1001   mutationAddConn2Val,...,
1002   mutationAddConn3Val,...,
1003   mutationAddConn1Val{:},...
1004   mutationAddConn2Val,...,
1005   mutationAddConn3Val,...,
1006   mutationAddConn1Val{:},...
1007   mutationAddConn2Val,...,
1008   mutationAddConn3Val,...,
1009   mutationAddConn1Val{:},...
1010   mutationAddConn2Val,...,
1011   mutationAddConn3Val,...,
1012   mutationAddConn1Val{:},...
1013   mutationAddConn2Val,...,
1014   mutationAddConn3Val,...,
1015   mutationAddConn1Val{:},...
1016   mutationAddConn2Val,...,
1017   mutationAddConn3Val,...,
1018   mutationAddConn1Val{:},...
1019   mutationAddConn2Val,...,
1020   mutationAddConn3Val,...,
1021   mutationAddConn1Val{:},...
1022   mutationAddConn2Val,...,
1023   mutationAddConn3Val,...,
1024   mutationAddConn1Val{:},...
1025   mutationAddConn2Val,...,
1026   mutationAddConn3Val,...,
1027   mutationAddConn1Val{:},...
1028   mutationAddConn2Val,...,
1029   mutationAddConn3Val,...,
1030   mutationAddConn1Val{:},...
1031   mutationAddConn2Val,...,
1032   mutationAddConn3Val,...,
1033   mutationAddConn1Val{:},...
1034   mutationAddConn2Val,...,
1035   mutationAddConn3Val,...,
1036   mutationAddConn1Val{:},...
1037   mutationAddConn2Val,...,
1038   mutationAddConn3Val,...,
1039   mutationAddConn1Val{:},...
1040   mutationAddConn2Val,...,
1041   mutationAddConn3Val,...,
1042   mutationAddConn1Val{:},...
1043   mutationAddConn2Val,...,
1044   mutationAddConn3Val,...,
1045   mutationAddConn1Val{:},...
1046   mutationAddConn2Val,...,
1047   mutationAddConn3Val,...,
1048   mutationAddConn1Val{:},...
1049   mutationAddConn2Val,...,
1050   mutationAddConn3Val,...,
1051   mutationAddConn1Val{:},...
1052   mutationAddConn2Val,...,
1053   mutationAddConn3Val,...,
1054   mutationAddConn1Val{:},...
1055   mutationAddConn2Val,...,
1056   mutationAddConn3Val,...,
1057   mutationAddConn1Val{:},...
1058   mutationAddConn2Val,...,
1059   mutationAddConn3Val,...,
1060   mutationAddConn1Val{:},...
1061   mutationAddConn2Val,...,
1062   mutationAddConn3Val,...,
1063   mutationAddConn1Val{:},...
1064   mutationAddConn2Val,...,
1065   mutationAddConn3Val,...,
1066   mutationAddConn1Val{:},...
1067   mutationAddConn2Val,...,
1068   mutationAddConn3Val,...,
1069   mutationAddConn1Val{:},...
1070   mutationAddConn2Val,...,
1071   mutationAddConn3Val,...,
1072   mutationAddConn1Val{:},...
1073   mutationAddConn2Val,...,
1074   mutationAddConn3Val,...,
1075   mutationAddConn1Val{:},...
1076   mutationAddConn2Val,...,
1077   mutationAddConn3Val,...,
1078   mutationAddConn1Val{:},...
1079   mutationAddConn2Val,...,
1080   mutationAddConn3Val,...,
1081   mutationAddConn1Val{:},...
1082   mutationAddConn2Val,...,
1083   mutationAddConn3Val,...,
1084   mutationAddConn1Val{:},...
1085   mutationAddConn2Val,...,
1086   mutationAddConn3Val,...,
1087   mutationAddConn1Val{:},...
1088   mutationAddConn2Val,...,
1089   mutationAddConn3Val,...,
1090   mutationAddConn1Val{:},...
1
```

```

487 Mutation...
488 evalFunHandle, ...
489 EvalFunParameters, ...
490 dateAndTime)
491
492 % Build best neural network
493 [neuralNet, nRelays] = build_neural_network(bestIndividual);
494
495 % Generate function
genFunctionMod(neuralNet, 'uav_best_nn_controller', ...
496 'MatrixOnly', 'yes', 'ShowLinks', 'yes');
497 pause(0.05);
498
499 % Number of previous states
500 nPreviousStates = nargin(funHandle) - nInputs;
501
502 % Post-process generated function
generated_nn_processing(...,
503 [strrep(pwd,'/','/'),'uav_best_nn_controller.m'], ...
504 'uav_best_nn_controller', nPreviousStates);
505
506 % View net
507 view(neuralNet);
508
509 % Terminate diary
510 diary('off');
511

```

B. Trajectory Definition

```

1 % This file contains the trajectories used for training of the neural network
2
3 if ~exist('iTrajSelection','var')
4 iTrajSelection = [1,2,6,8,9,14,16];
5 end
6
7 %X Trajectory 1 - zDesired - zInitial = -90 m
8
9 iTraj = 1;
10 nPoints = 1000;
11 trajectoryType = 'Linear';
12 ts = 0; % Start time (s)
13 qs = -90; % Start position (m)
14 tf = 15; % Finish time (s)
15 qf = -90; % Finish position (m)
16 [zDesiredTime, zDesired] = trajectory_generation(nPoints, trajectoryType, ...
17 ts, qs, tf, qf);
18 EvalFunParameters.Trajectory(iTraj).XDesired.t = zDesiredTime;
19 EvalFunParameters.Trajectory(iTraj).XDesired.val = zeros(1,nPoints);
20 EvalFunParameters.Trajectory(iTraj).XDesired.tol = [-30,+30]*MM_2_M;
21 EvalFunParameters.Trajectory(iTraj).YDesired.t = zDesiredTime;
22 EvalFunParameters.Trajectory(iTraj).YDesired.val = zeros(1,nPoints);
23 EvalFunParameters.Trajectory(iTraj).YDesired.tol = [-30,+30]*MM_2_M;
24 EvalFunParameters.Trajectory(iTraj).ZDesired.t = zDesiredTime;
25 EvalFunParameters.Trajectory(iTraj).ZDesired.val = [-30,+30]*MM_2_M;
26 EvalFunParameters.Trajectory(iTraj).ZDesired.tol = zDesiredTime;
27 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
28 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
29 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
30 EvalFunParameters.Trajectory(iTraj).ThetaDesired.t = zDesiredTime;
31 EvalFunParameters.Trajectory(iTraj).ThetaDesired.val = zeros(1,nPoints);
32 EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
33 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
34 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
35 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
36 EvalFunParameters.Trajectory(iTraj).InitCond.xPInitial = zeros(3,1);
37 EvalFunParameters.Trajectory(iTraj).InitCond.omegabInitial = zeros(4,1);
38 EvalFunParameters.Trajectory(iTraj).InitCond.iainit = zeros(4,1);
39 EvalFunParameters.Trajectory(iTraj).InitCond.iadelay = zeros(4,1);
40 EvalFunParameters.Trajectory(iTraj).InitCond.vptInitial = zeros(3,1);
41 EvalFunParameters.Trajectory(iTraj).InitCond.vptDelay = zeros(3,1);
42 EvalFunParameters.Trajectory(iTraj).InitCond.xpInitial = [0;0;-90];
43 EvalFunParameters.Trajectory(iTraj).InitCond.xpDelay = zeros(3,1);
44 EvalFunParameters.Trajectory(iTraj).InitCond.omegabInitial = zeros(3,1);
45 EvalFunParameters.Trajectory(iTraj).InitCond.eulerZYXInitial = zeros(3,1);
46 EvalFunParameters.Trajectory(iTraj).InitCond.eulerZYXDelay = zeros(3,1);
47 EvalFunParameters.Trajectory(iTraj).maxViolationX = 1.5; % 300
48 EvalFunParameters.Trajectory(iTraj).maxViolationY = 1.5; % 300
49 EvalFunParameters.Trajectory(iTraj).maxViolationZ = 2; % 300
50 EvalFunParameters.Trajectory(iTraj).maxViolationPsi = 0.3; % 150
51 EvalFunParameters.Trajectory(iTraj).maxViolationTheta = 0.3; % 150
52 EvalFunParameters.Trajectory(iTraj).maxViolationPhi = 0.3; % 150
53 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi = 1;
54 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
55 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
56 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
57 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi = 0.25;
58 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
59 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi = 0.25;
60
61 %X Trajectory 2 - zDesired - zInitial = -50 m
62
63 iTraj = 2;
64 nPoints = 1000;
65 trajectoryType = 'Linear';
66 ts = 0; % Start time (s)
67 qs = -50; % Start position (m)
68 tf = 15; % Finish time (s)
69 qf = -50; % Finish position (m)
70 [zDesiredTime, zDesired] = trajectory_generation(nPoints, trajectoryType, ...
71 ts, qs, tf, qf);
72 EvalFunParameters.Trajectory(iTraj).XDesired.t = zDesiredTime;
73 EvalFunParameters.Trajectory(iTraj).XDesired.val = zeros(1,nPoints);
74 EvalFunParameters.Trajectory(iTraj).XDesired.tol = [-30,+30]*MM_2_M;
75 EvalFunParameters.Trajectory(iTraj).YDesired.t = zDesiredTime;
76 EvalFunParameters.Trajectory(iTraj).YDesired.val = zeros(1,nPoints);
77 EvalFunParameters.Trajectory(iTraj).YDesired.tol = [-30,+30]*MM_2_M;
78 EvalFunParameters.Trajectory(iTraj).ZDesired.t = zDesiredTime;
79 EvalFunParameters.Trajectory(iTraj).ZDesired.val = [-30,+30]*MM_2_M;
80 EvalFunParameters.Trajectory(iTraj).ZDesired.tol = zDesiredTime;
81 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
82 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
83 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-30,+30]*MM_2_M;
84 EvalFunParameters.Trajectory(iTraj).ZDesired.t = zDesiredTime;
85 EvalFunParameters.Trajectory(iTraj).ZDesired.val = [-30,+30]*MM_2_M;
86 EvalFunParameters.Trajectory(iTraj).ZDesired.tol = zDesiredTime;
87 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
88 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
89 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
90 EvalFunParameters.Trajectory(iTraj).ThetaDesired.t = zDesiredTime;
91 EvalFunParameters.Trajectory(iTraj).ThetaDesired.val = zeros(1,nPoints);
92 EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
93 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
94 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
95 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
96 EvalFunParameters.Trajectory(iTraj).InitCond.omegalInitial = 0*ones(4,1);
97 EvalFunParameters.Trajectory(iTraj).InitCond.omegabInitial = zeros(4,1);
98 EvalFunParameters.Trajectory(iTraj).InitCond.iainit = zeros(4,1);
99 EvalFunParameters.Trajectory(iTraj).InitCond.iadelay = zeros(4,1);
100 EvalFunParameters.Trajectory(iTraj).InitCond.vptInitial = zeros(3,1);
101 EvalFunParameters.Trajectory(iTraj).InitCond.vptDelay = zeros(3,1);
102 EvalFunParameters.Trajectory(iTraj).InitCond.xpInitial = [0;0;-60];
103 EvalFunParameters.Trajectory(iTraj).InitCond.xpDelay = zeros(3,1);
104 EvalFunParameters.Trajectory(iTraj).InitCond.omegabInitial = zeros(3,1);
105 EvalFunParameters.Trajectory(iTraj).InitCond.omegabDelay = zeros(3,1);
106 EvalFunParameters.Trajectory(iTraj).InitCond.eulerZYXInitial = zeros(3,1);
107 EvalFunParameters.Trajectory(iTraj).InitCond.eulerZYXDelay = zeros(3,1);
108 EvalFunParameters.Trajectory(iTraj).maxViolationX = 1.5;
109 EvalFunParameters.Trajectory(iTraj).maxViolationY = 1.5;
110 EvalFunParameters.Trajectory(iTraj).maxViolationZ = 2;
111 EvalFunParameters.Trajectory(iTraj).maxViolationPsi = 0.3;
112 EvalFunParameters.Trajectory(iTraj).maxViolationTheta = 0.3;
113 EvalFunParameters.Trajectory(iTraj).maxViolationPhi = 0.3;
114 EvalFunParameters.Trajectory(iTraj).violationFactorX = 3;
115 EvalFunParameters.Trajectory(iTraj).violationFactorY = 3;
116 EvalFunParameters.Trajectory(iTraj).violationFactorZ = 5;
117 EvalFunParameters.Trajectory(iTraj).violationFactorPsi = 2;
118 EvalFunParameters.Trajectory(iTraj).violationFactorTheta = 2;
119 EvalFunParameters.Trajectory(iTraj).violationFactorPhi = 2;
120 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
121 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
122 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
123 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi = 0.25;
124 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
125 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi = 0.25;
126
127 %X Trajectory 3 - zDesired - zInitial = -25 m
128 iTraj = 3;
129 nPoints = 1000;
130 trajectoryType = 'Linear';
131 ts = 0; % Start time (s)
132 qs = -25; % Start position (m)
133 tf = 15; % Finish time (s)
134 qf = -25; % Finish position (m)
135 [zDesiredTime, zDesired] = trajectory_generation(nPoints, trajectoryType, ...
136 ts, qs, tf, qf);
137 EvalFunParameters.Trajectory(iTraj).XDesired.t = zDesiredTime;
138 EvalFunParameters.Trajectory(iTraj).XDesired.val = zeros(1,nPoints);
139 EvalFunParameters.Trajectory(iTraj).XDesired.tol = [-30,+30]*MM_2_M;
140 EvalFunParameters.Trajectory(iTraj).YDesired.t = zDesiredTime;
141 EvalFunParameters.Trajectory(iTraj).YDesired.val = zeros(1,nPoints);
142 EvalFunParameters.Trajectory(iTraj).YDesired.tol = [-30,+30]*MM_2_M;
143 EvalFunParameters.Trajectory(iTraj).ZDesired.t = zDesiredTime;
144 EvalFunParameters.Trajectory(iTraj).ZDesired.val = zeros(1,nPoints);
145 EvalFunParameters.Trajectory(iTraj).ZDesired.tol = zDesiredTime;
146 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
147 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-30,+30]*MM_2_M;
148 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
149 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
150 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
151 EvalFunParameters.Trajectory(iTraj).ThetaDesired.val = zeros(1,nPoints);
152 EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
153 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
154 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
155 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
156 EvalFunParameters.Trajectory(iTraj).InitCond.omegalInitial = 0*ones(4,1);
157 EvalFunParameters.Trajectory(iTraj).InitCond.omegabInitial = zeros(4,1);
158 EvalFunParameters.Trajectory(iTraj).InitCond.iainit = zeros(4,1);
159 EvalFunParameters.Trajectory(iTraj).InitCond.iadelay = zeros(4,1);
160 EvalFunParameters.Trajectory(iTraj).InitCond.vptInitial = zeros(3,1);
161 EvalFunParameters.Trajectory(iTraj).InitCond.vptDelay = zeros(3,1);
162 EvalFunParameters.Trajectory(iTraj).InitCond.xpInitial = [0;0;-25];
163 EvalFunParameters.Trajectory(iTraj).InitCond.xpDelay = zeros(3,1);
164 EvalFunParameters.Trajectory(iTraj).InitCond.omegabInitial = zeros(3,1);
165 EvalFunParameters.Trajectory(iTraj).InitCond.omegabDelay = zeros(3,1);
166 EvalFunParameters.Trajectory(iTraj).InitCond.eulerZYXInitial = zeros(3,1);
167 EvalFunParameters.Trajectory(iTraj).InitCond.eulerZYXDelay = zeros(3,1);
168 EvalFunParameters.Trajectory(iTraj).maxViolationX = 1.5;
169 EvalFunParameters.Trajectory(iTraj).maxViolationY = 1.5;
170 EvalFunParameters.Trajectory(iTraj).maxViolationZ = 2;
171 EvalFunParameters.Trajectory(iTraj).maxViolationPsi = 0.3;
172 EvalFunParameters.Trajectory(iTraj).maxViolationTheta = 0.3;
173 EvalFunParameters.Trajectory(iTraj).maxViolationPhi = 0.3;
174 EvalFunParameters.Trajectory(iTraj).violationFactorX = 3;
175 EvalFunParameters.Trajectory(iTraj).violationFactorY = 3;
176 EvalFunParameters.Trajectory(iTraj).violationFactorZ = 5;
177 EvalFunParameters.Trajectory(iTraj).violationFactorPsi = 2;
178 EvalFunParameters.Trajectory(iTraj).violationFactorTheta = 2;
179 EvalFunParameters.Trajectory(iTraj).violationFactorPhi = 2;
180 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
181 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
182 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
183 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi = 0.25;
184 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
185 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi = 0.25;
186
187 %X Trajectory 4 - zDesired - zInitial = 0 m
188
189 iTraj = 4;
190 nPoints = 1000;
191 trajectoryType = 'Linear';
192 ts = 0; % Start time (s)
193 qs = 0; % Start position (m)
194 tf = 15; % Finish time (s)
195 qf = 0; % Finish position (m)
196 [zDesiredTime, zDesired] = trajectory_generation(nPoints, trajectoryType, ...
197 ts, qs, tf, qf);
198 EvalFunParameters.Trajectory(iTraj).XDesired.t = zDesiredTime;
199 EvalFunParameters.Trajectory(iTraj).XDesired.val = zeros(1,nPoints);
200 EvalFunParameters.Trajectory(iTraj).XDesired.tol = [-30,+30]*MM_2_M;
201 EvalFunParameters.Trajectory(iTraj).YDesired.t = zDesiredTime;
202 EvalFunParameters.Trajectory(iTraj).YDesired.val = zeros(1,nPoints);
203 EvalFunParameters.Trajectory(iTraj).YDesired.tol = [-30,+30]*MM_2_M;
204 EvalFunParameters.Trajectory(iTraj).ZDesired.t = zDesiredTime;
205 EvalFunParameters.Trajectory(iTraj).ZDesired.val = zeros(1,nPoints);
206 EvalFunParameters.Trajectory(iTraj).ZDesired.tol = zDesiredTime;
207 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
208 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
209 EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
210 EvalFunParameters.Trajectory(iTraj).ThetaDesired.t = zDesiredTime;
211 EvalFunParameters.Trajectory(iTraj).ThetaDesired.val = zeros(1,nPoints);
212 EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
213 EvalFunParameters.Trajectory(iTraj).PhiDesired.t = zDesiredTime;
214 EvalFunParameters.Trajectory(iTraj).PhiDesired.val = zeros(1,nPoints);
215 EvalFunParameters.Trajectory(iTraj).PhiDesired.tol = [-3.5,+3.5]*DEG_2_RAD;
216 EvalFunParameters.Trajectory(iTraj).InitCond.omegalInitial = 0*ones(4,1);
217 EvalFunParameters.Trajectory(iTraj).InitCond.omegabInitial = zeros(4,1);
218 EvalFunParameters.Trajectory(iTraj).InitCond.iainit = zeros(4,1);
219 EvalFunParameters.Trajectory(iTraj).InitCond.iadelay = zeros(4,1);
220 EvalFunParameters.Trajectory(iTraj).InitCond.vptInitial = zeros(3,1);
221 EvalFunParameters.Trajectory(iTraj).InitCond.vptDelay = zeros(3,1);
222 EvalFunParameters.Trajectory(iTraj).InitCond.xpInitial = [0;0;0];
223 EvalFunParameters.Trajectory(iTraj).InitCond.xpDelay = zeros(3,1);
224 EvalFunParameters.Trajectory(iTraj).InitCond.omegabInitial = zeros(3,1);
225 EvalFunParameters.Trajectory(iTraj).InitCond.omegabDelay = zeros(3,1);
226 EvalFunParameters.Trajectory(iTraj).InitCond.eulerZYXInitial = zeros(3,1);
227 EvalFunParameters.Trajectory(iTraj).InitCond.eulerZYXDelay = zeros(3,1);
228 EvalFunParameters.Trajectory(iTraj).maxViolationX = 1.5;
229 EvalFunParameters.Trajectory(iTraj).maxViolationY = 1.5;
230 EvalFunParameters.Trajectory(iTraj).maxViolationZ = 2;
231 EvalFunParameters.Trajectory(iTraj).maxViolationPsi = 0.3;
232 EvalFunParameters.Trajectory(iTraj).maxViolationTheta = 0.3;
233 EvalFunParameters.Trajectory(iTraj).maxViolationPhi = 0.3;
234 EvalFunParameters.Trajectory(iTraj).violationFactorX = 3;
235 EvalFunParameters.Trajectory(iTraj).violationFactorY = 3;
236 EvalFunParameters.Trajectory(iTraj).violationFactorZ = 5;
237 EvalFunParameters.Trajectory(iTraj).violationFactorPsi = 2;
238 EvalFunParameters.Trajectory(iTraj).violationFactorTheta = 2;
239 EvalFunParameters.Trajectory(iTraj).violationFactorPhi = 2;
240 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
241 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
242 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
243 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi = 0.25;
244 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
245 EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi = 0.25;

```


C. NEAT

```

1 %X neat() (main)
2 function [Population, bestIndividual, varargout] = neat...
3 nMaxGeneration, ...
4 loadGene, ...
5 geneFile, ...
6 loadFlag, ...
7 saveFlag, ...
8 populationSize, ...
9 nInputNodes, ...
10 nOutputNodes, ...
11 initialConnectedInputNodes, ...
12 fitnessLimit, ...
13 Speciation, ...
14 Initial, ...
15 Stagnation, ...
16 Refocus, ...
17 Selection, ...
18 Crossover, ...
19 Mutation, ...
20 evalFunHandle, ...
21 EvalFunParameters, ...
22 dateAndTime, ...
23 varargin)
24
25 %X Initialization
26
27 % Timing
28 tic;
29
30 % Initialize vectors
31 nAverageNonDisabledConnections = [];
32 nAverageHiddenNodes = [];
33 maxOverallFitness = [];
34 meanOverallFitness = [];
35 meanRecurrentConnection = [];
36 nIndPerSpecies = [];
37
38 % Initialize Speciation structure. It will contain various information on
% single species. This data will be used for fitness sharing, reproduction,
39 % and for visualisation purposes.
40 SpeciesRecord(1).id = 0; % Consecutive species ID's
41 SpeciesRecord(1).nIndividuals = 0; % Number of individuals in species
42 % generationRecord matrix will be 4 rows by (number of generations
43 % existent) columns, will contain (from top to bottom):
44 % - Number of generation
45 % - Mean raw fitness
46 % - Mean raw fitness
47 % - Index of individual in population which has produced max raw fitness
48 SpeciesRecord(1).generationRecord = [];
49
50 if loadFlag == 0 && loadGene == 0
51
52 % Call function to create Initial population
53 % for information about the make-up of the population structure and the
54 % innovationRecord, look at initialize_population()
55 [Population, innovationRecord] = initialize_population(...%
56 populationSize, nInputNodes, nOutputNodes, initialConnectedInputNodes);
57
58 % Initial speciation
59 [Population, SpeciesRecord] = ...
60 initial_speciation(initialConnectedInputNodes, Population, nOutputNodes, ...
61 SpeciesRecord, Speciation, loadGene);
62
63 % Generation counter
64 iGeneration = 1;
65
66 elseif loadFlag == 0 && loadGene == 1
67
68 % Load gene
69 TmpStruct = load(geneFile,'Population');
70 PopulationTmp = TmpStruct.Population;
71 if numel([PopulationTmp().fitness]) == 1
72 startNodeGenes = PopulationTmp.nodeGenes;
73 startConnectionGenes = PopulationTmp.connectionGenes;
74 else
75 tmpFitness = [PopulationTmp().fitness];
76 [~, iTopIndividual] = max(tmpFitness);
77 bestIndividual = PopulationTmp(iTopIndividual);
78 startNodeGenes = bestIndividual.nodeGenes;
79 startConnectionGenes = bestIndividual.connectionGenes;
80 end
81
82 % Initialize population with gene
83 [Population, innovationRecord] = initialize_population_custom_gene(...%
84 populationSize, ...
85 startNodeGenes, ...
86 startConnectionGenes);
87
88 % Initial speciation
89 [Population, SpeciesRecord] = ...
90 initial_speciation(initialConnectedInputNodes, Population, nOutputNodes, ...
91 SpeciesRecord, Speciation, loadGene);
92
93 % Generation counter
94 iGeneration = 1;
95
96 else
97
98 % Backup mutation scheduling
99 probabilityAddNodeSchedulingTmp = Mutation.probabilityAddNodeScheduling;
100 probabilityAddConnectionSchedulingTmp = Mutation.probabilityAddConnectionScheduling;
101 if true
102 weightCapTmp = Mutation.weightCap;
103 weightRangeTmp = Mutation.weightRange;
104 end
105
106 % Start with saved version of evolution
107 load('neatsave.mat');
108
109 % Restore mutation scheduling
110 Mutation.probabilityAddNodeScheduling = probabilityAddNodeSchedulingTmp;
111 Mutation.probabilityAddConnectionScheduling = probabilityAddConnectionSchedulingTmp;
112 if true
113 Mutation.weightCap = weightCapTmp;
114 Mutation.weightRange = weightRangeTmp;
115 end
116
117 if Mutation.probabilityAddNodeScheduling() == -1
118 Mutation.probabilityAddNode = Mutation.probabilityAddNodeScheduling(2,1);
119 end
120
121 if Mutation.probabilityAddConnectionScheduling() == -1
122 Mutation.probabilityAddConnection = Mutation.probabilityAddConnectionScheduling(2,1);
123 end
124
125 if Mutation.probabilityAddConnectionScheduling() == -1
126 Mutation.probabilityAddConnection = Mutation.probabilityAddConnectionScheduling(2,1);
127 end
128
129 %X Generational Loop
130 print_generation(iGeneration);
131 flagSolution = 0;
132 while (iGeneration < nMaxGeneration) && (flagSolution == 0)
133
134
135
136 % Backup copies of current generation
137 if saveFlag == 1
138 save('neatsave.mat','Population','iGeneration','innovationRecord',...
139 'SpeciesRecord','nAverageNonDisabledConnections','nAverageHiddenNodes',...
140 'maxOverallFitness','dateAndTime','Mutation','nAverageDisabledConnections',...
141 'nSpeciesArray','meanOverallFitness','meanRecurrentConnection','nIndPerSpecies');
142 if exist('figHandle','var')
143 if ishandle(figHandle)
144 savefig(figHandle,'neatfig.fig');
145 end
146 end
147 end
148
149 % Call evaluation function (in this case XOR), fitnesses of individuals
150 % will be stored in Population().fitness.
151 % IMPORTANT: reproduction assumes an (all positive!) evaluation function
152 % where a higher value means better fitness (in other words, the
153 % only value used between 0 and +Inf)
154 % Population = xor_experiment(Population);
155 % Population = fulladder_experiment(Population);
156 % Population = evalFunHandle(Population, fitnessLimit);
157 if isempty(EvalFunParameters)
158 Population = evalFunHandle(Population, fitnessLimit);
159 else
160 nSimPoints = 5000;
161 SimData = initialize_sim_data_structure(EvalFunParameters, populationSize, ...
162 nSimPoints);
163 EvalFunParameters.iGeneration = iGeneration;
164 [Population, SimData] = evalFunHandle(Population, EvalFunParameters, ...
165 fitnessLimit, SimData);
166 end
167
168 % Check for stagnation
169 [maxFitnessesCurrentGeneration, SpeciesRecord] = ...
170 stagnation(SpeciesRecord, Population, Stagnation, iGeneration);
171
172 % Check for refocus
173 [SpeciesRecord] = ...
174 refocus(maxFitnessesCurrentGeneration, SpeciesRecord, Refocus);
175
176 % Save current generation
177 folderName = 'neat_data';
178 if ~exist(folderName,'dir')
179 mkdir(folderName);
180 end
181 save([folderName '/neatsave_','dateAndTime','_',num2str(iGeneration,'X03.of'),'.mat'],'...
182 'Population','iGeneration','innovationRecord','SpeciesRecord');
183
184 % Compute stats
185 [nAverageNonDisabledConnections, nAverageHiddenNodes, maxFitness, ...
186 maxOverallFitness, nAverageDisabledConnections, ...
187 nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecies] = ...
188 compute_stats(Population, populationSize, iGeneration, ...
189 SpeciesRecord, nAverageNonDisabledConnections, nAverageHiddenNodes, ...
190 maxOverallFitness, nAverageDisabledConnections, ...
191 nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecies);
192
193 % Create figure handle and axes if don't exist or have been deleted
194 if exist('figHandle','var')
195 [figHandle, AxHandle] = neat_plot_helper(loadFlag, figHandle, AxHandle);
196 figHandle.CloseRequestFcn = @closereqmod;
197 figHandle.Resize = 'off';
198 else
199 [figHandle, AxHandle] = neat_plot_helper(loadFlag);
200 figHandle.CloseRequestFcn = @closereqmod;
201 figHandle.Resize = 'off';
202 end
203
204 % Visualisation fitness & species
205 visualization(figHandle, AxHandle, nAverageNonDisabledConnections, ...
206 nAverageHiddenNodes, maxOverallFitness, nAverageDisabledConnections, ...
207 nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecies);
208
209 % Find best individual of this generation
210 tmpFitness = [Population().fitness];
211 [~, iTopIndividual] = max(tmpFitness);
212 bestIndividual = Population(iTopIndividual);
213
214 % Visualisation of simulation results of the best individual
215 if exist('figHandleSim','var')
216 if ~exist('figHandleSimError','var'); figHandleSim = []; end
217 if ~exist('figHandleSimError','var'); figHandleSimError = []; end
218 [figHandleSim, figHandleSimError] = ...
219 nn.visualization(figHandleSim, figHandleSimError, SimData, ...
220 iTopIndividual, EvalFunParameters);
221 figHandleSim.CloseRequestFcn = @closereqmod;
222 figHandleSim.Error.CloseRequestFcn = @closereqmod;
223 figHandleSim.Resize = 'off';
224 figHandleSimError.Resize = 'off';
225 end
226
227 % Visualisation of neural network of the best individual
228 if exist('figHandleNeuralNet','var')
229 figHandleNeuralNet = nn.visualization(Population, figHandleNeuralNet);
230 figHandleNeuralNet.CloseRequestFcn = @closereqmod;
231 figHandleNeuralNet.Resize = 'off';
232 else
233 figHandleNeuralNet = nn.visualization(Population);
234 figHandleNeuralNet.CloseRequestFcn = @closereqmod;
235 figHandleNeuralNet.Resize = 'off';
236 end
237
238 % Check for solution
239 flagSolution = solution_check(maxFitness, fitnessLimit, flagSolution);
240
241 % We call reproduce if there's not solution yet
242 if flagSolution == 0
243 % Call reproduction function with parameters, current population
244 % and species record, returns new population, new species record
245 % and new innovation record
246 [Population, SpeciesRecord, innovationRecord] = ...
247 reproduce(Population, SpeciesRecord, innovationRecord, Initial, ...
248 Selection, Crossover, Mutation, Speciation, iGeneration, populationSize);
249 % Display time
250 display_elapsed_time(toc());
251
252 % Increment generational counter
253 iGeneration = iGeneration + 1;
254
255 % Print generation to command window
256 print_generation(iGeneration);
257
258 % Reduce mutation probabilities (if shrink rate > 0)
259 Mutation = shrink_mutation(Mutation, iGeneration);
260
261
262 end
263
264 end
265
266 %X initialize_sim_data_structure()
267 function SimData = ...
268 initialize_sim_data_structure(EvalFunParameters, populationSize, nSimPoints)
269
270 allrayTm = cell(1,numel(EvalFunParameters.Tau));
271 [cellarrayTm{:}] = deal(zeros(nSimPoints,1));
272 [SimData(:,populationSize).t] = deal(cellarrayTm);
273 [SimData(:,populationSize).xDesired] = deal(cellarrayTm);
274 [SimData(:,populationSize).zDesired] = deal(cellarrayTm);
275 [SimData(:,populationSize).zDesired] = deal(cellarrayTm);
276 [SimData(:,populationSize).psiDesired] = deal(cellarrayTm);
277 [SimData(:,populationSize).thetaDesired] = deal(cellarrayTm);

```

```

278 [SimData(1:populationSize).phiDesired] = deal(cellArrayTmp);
279 [SimData(1:populationSize).x] = deal(cellArrayTmp);
280 [SimData(1:populationSize).y] = deal(cellArrayTmp);
281 [SimData(1:populationSize).z] = deal(cellArrayTmp);
282 [SimData(1:populationSize).psi] = deal(cellArrayTmp);
283 [SimData(1:populationSize).theta] = deal(cellArrayTmp);
284 [SimData(1:populationSize).phi] = deal(cellArrayTmp);
285 [SimData(1:populationSize).Va1] = deal(cellArrayTmp);
286 [SimData(1:populationSize).Va2] = deal(cellArrayTmp);
287 [SimData(1:populationSize).Va3] = deal(cellArrayTmp);
288 [SimData(1:populationSize).Va4] = deal(cellArrayTmp);
289
290 end
291
292 % display_elapsed_time()
293 function display_elapsed_time(elapsedTime)
294
295 % elapsedTime = toc;
296 elapsedTimeHours = floor(elapsedTime/3600);
297 elapsedTimeMins = floor((elapsedTime - 3600*elapsedTimeHours)/60);
298 elapsedTimeSecs = floor((elapsedTime - 3600*elapsedTimeHours - 60*elapsedTimeMins));
299 fprintf('Elapsed time is %02.0f%02.0f%02.0f.%03d\n',...
300 elapsedTimeHours, elapsedTimeMins, elapsedTimeSecs);
301
302 end
303
304 %% sim_visualization()
305 function [figHandleSim, figHandleSimError] = ...
306 sim_visualization(figHandleSim, figHandleSimError, SimData, iTopIndividual, ...
307 EvalFunParameters)
308
309 if exist('SimData','var')
310
311 if isempty(figHandleSim) & ~isempty(figHandleSimError)
312 if ishandle(figHandleSim) & ishandle(figHandleSimError)
313 clf(figHandleSim);
314 clf(figHandleSimError);
315 else
316 if ishandle(figHandleSim) || ~ishandle(figHandleSimError)
317 if ishandle(figHandleSim); close(figHandleSim); end;
318 if ishandle(figHandleSimError); close(figHandleSimError); end;
319 delete figHandleSim;
320 end
321 end
322
323 nTraj = numel(EvalFunParameters.Trajectory);
324 [Data(1:nTraj).t] = deal(SimData(iTopIndividual).t(:));
325 [Data(1:nTraj).xDesired] = deal(SimData(iTopIndividual).xDesired(:));
326 [Data(1:nTraj).yDesired] = deal(SimData(iTopIndividual).yDesired(:));
327 [Data(1:nTraj).zDesired] = deal(SimData(iTopIndividual).zDesired(:));
328 [Data(1:nTraj).psiDesired] = deal(SimData(iTopIndividual).psiDesired(:));
329 [Data(1:nTraj).thetaDesired] = deal(SimData(iTopIndividual).thetaDesired(:));
330 [Data(1:nTraj).phiDesired] = deal(SimData(iTopIndividual).phiDesired(:));
331 [Data(1:nTraj).x] = deal(SimData(iTopIndividual).x(:));
332 [Data(1:nTraj).y] = deal(SimData(iTopIndividual).y(:));
333 [Data(1:nTraj).z] = deal(SimData(iTopIndividual).z(:));
334 [Data(1:nTraj).psi] = deal(SimData(iTopIndividual).psi(:));
335 [Data(1:nTraj).theta] = deal(SimData(iTopIndividual).theta(:));
336 [Data(1:nTraj).phi] = deal(SimData(iTopIndividual).phi(:));
337 [Data(1:nTraj).Va1] = deal(SimData(iTopIndividual).Va1(:));
338 [Data(1:nTraj).Va2] = deal(SimData(iTopIndividual).Va2(:));
339 [Data(1:nTraj).Va3] = deal(SimData(iTopIndividual).Va3(:));
340 [Data(1:nTraj).Va4] = deal(SimData(iTopIndividual).Va4(:));
341
342 customLegend = cell(1,nTraj);
343 for jTraj = 1:nTraj; customLegend{jTraj} = ['Traj ',num2str(jTraj)]; end;
344
345 if isempty(figHandleSim) & ~isempty(figHandleSimError) & ...
346 ishandle(figHandleSim) & ishandle(figHandleSimError)
347 [figHandleSim, figHandleSimError] =
348 plot_uav_sim(Data, true, false, false, true, false, ...
349 false, customLegend, true, figHandleSim, figHandleSimError);
350 drawnow;
351 else
352 [figHandleSim, figHandleSimError] = ...
353 plot_uav_sim(Data, true, false, false, true, false, ...
354 false, customLegend, true);
355 drawnow;
356 end
357
358 end
359
360 end
361
362 %% nn_visualization()
363 function figHandleNeuralNet = nn_visualization(Population, varargin)
364
365 if nargin == 2
366 figHandleNeuralNet = varargin{1};
367 end
368
369 if exist('figHandleNeuralNet','var') & ishandle(figHandleNeuralNet)
370 clf(figHandleNeuralNet);
371 end
372
373 if exist('figHandleNeuralNet','var') & ishandle(figHandleNeuralNet)
374 figHandleNeuralNet = plot_neural_network([], ...
375 [], [], [], 1, true, false, Population, figHandleNeuralNet);
376 drawnow;
377 else
378 figHandleNeuralNet = plot_neural_network([], ...
379 [], [], [], 1, true, false, Population);
380 drawnow;
381 end
382
383 end
384
385 %% neat_plot_helper()
386 function [figHandle, AxHandle] = neat_plot_helper(loadFlag, varargin)
387
388 % This function simply manages the figure and axes of neat plot.
389
390 if nargin == 3
391 figHandle = varargin{1};
392 AxHandle = varargin{2};
393 end
394
395 marginSubPlot = 0.10;
396 subplotSettings = {marginSubPlot, 'NextPlot', 'replacechildren', 'Tag'};
397
398 if ~exist('figHandle','var') & loadFlag == 1
399
400 figHandle = createFigure(800, 700, 'on');
401 AxHandle.subplot1 = subplot(tight(3,3,[1,2], subplotSettings{:}, 'subPlot1'));
402 AxHandle.subplot2 = subplot(tight(3,3,3, subplotSettings{:}, 'subPlot2'));
403 AxHandle.subplot3 = subplot(tight(3,3,4, subplotSettings{:}, 'subPlot3'));
404 AxHandle.subplot4 = subplot(tight(3,3,5, subplotSettings{:}, 'subPlot4'));
405 AxHandle.subplot5 = subplot(tight(3,3,6, subplotSettings{:}, 'subPlot5'));
406 AxHandle.subplot6 = subplot(tight(3,3,7, subplotSettings{:}, 'subPlot6'));
407 AxHandle.subplot7 = subplot(tight(3,3,8, subplotSettings{:}, 'subPlot7'));
408 AxHandle.subplot8 = subplot(tight(3,3,9, subplotSettings{:}, 'subPlot8'));
409
410 elseif ~exist('figHandle','var') & loadFlag == 1
411
412 f1Handle = openfig('heatfig_1');
413 AxHandle.subplot1 = findobj(f1Handle, 'Type', 'axes', 'Tag', 'subPlot1');
414 AxHandle.subplot2 = findobj(f1Handle, 'Type', 'axes', 'Tag', 'subPlot2');
415 AxHandle.subplot3 = findobj(f1Handle, 'Type', 'axes', 'Tag', 'subPlot3');
416 AxHandle.subplot4 = findobj(f1Handle, 'Type', 'axes', 'Tag', 'subPlot4');
417 AxHandle.subplot5 = findobj(f1Handle, 'Type', 'axes', 'Tag', 'subPlot5');
418 AxHandle.subplot6 = findobj(f1Handle, 'Type', 'axes', 'Tag', 'subPlot6');
419 AxHandle.subplot7 = findobj(f1Handle, 'Type', 'axes', 'Tag', 'subPlot7');
420
421 AxHandle.subPlot8 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot8');
422
423
424 % In case the figure has been closed
425 if exist('figHandle','var') & ~ishandle(figHandle)
426
427 fprintf('\nNEAT plot window has been closed. It will be reopened the next generation.\n');
428 delete AxHandle; clear AxHandle;
429 delete figHandle; clear figHandle;
430 pop;
431 figHandle = createFigure(800, 700, 'on');
432 drawnow;
433 set(0, 'currentfigure', figHandle);
434 AxHandle.subplot1 = subplot(tight(3,3,[1,2], subplotSettings{:}, 'subPlot1'));
435 set(0, 'currentfigure', figHandle);
436 AxHandle.subplot2 = subplot(tight(3,3,3, subplotSettings{:}, 'subPlot2'));
437 set(0, 'currentfigure', figHandle);
438 AxHandle.subplot3 = subplot(tight(3,3,4, subplotSettings{:}, 'subPlot3'));
439 set(0, 'currentfigure', figHandle);
440 AxHandle.subplot4 = subplot(tight(3,3,5, subplotSettings{:}, 'subPlot4'));
441 set(0, 'currentfigure', figHandle);
442 AxHandle.subplot5 = subplot(tight(3,3,6, subplotSettings{:}, 'subPlot5'));
443 set(0, 'currentfigure', figHandle);
444 AxHandle.subplot6 = subplot(tight(3,3,7, subplotSettings{:}, 'subPlot6'));
445 set(0, 'currentfigure', figHandle);
446 AxHandle.subplot7 = subplot(tight(3,3,8, subplotSettings{:}, 'subPlot7'));
447 set(0, 'currentfigure', figHandle);
448 AxHandle.subplot8 = subplot(tight(3,3,9, subplotSettings{:}, 'subPlot8'));
449
450 end
451
452 end
453
454 %% initialize_population()
455 function [Population,innovationRecord] = initialize_population...
456 populationSize,...
457 nInputNodes...
458 nOutputNodes...
459 initialConnectedInputNodes
460
461 % nodeGenes is array 4 rows * (nInputNodes + nOutputNodes + ...
462 % hidden-nodes (not existent in initial population) + 1 (bias-node)) columns
463 % nodeGenes contains:
464 % - Consecutive node ID's (upper row)
465 % - Node type (lower row) 1 = input 2 = output 3 = hidden 4 = bias
466 % - Node input state
467 % - Node output state (used for evaluation, all input states zero
468 % initially, except bias node, which is always 1)
469 %
470 % connectionGenes is array 5 rows * nConnections columns
471 % from top to bottom, those five rows contain:
472 % - Connection number
473 % - Connection from
474 % - Connection to
475 % - Weight
476 % - Enable bit
477 % The rest of the elements in the structure for an individual should be
478 % self-explanatory
479 %
480 % innovationRecord tracks innovations in a 5 rows by (number of
481 % innovations) columns matrix, contains:
482 % - innovation number
483 % - connectedFrom for this innovation
484 % - connectedTo for this innovation
485 % - The node (if it is a new node mutation, then this node will appear in
486 % the 4th row when it is first connected. There will always be two
487 % innovations with one node mutation, since there is a connection to and
488 % from the new node. In the initial population, this will be abbreviated to
489 % the node with the highest number appearing in the last column of the record)
490 % since only this is needed as starting point for the rest of the algorithm
491 % 5th row is generation this innovation occurred (generation is assumed
492 % to be zero for the innovations in the initial population)
493
494 % Compute number and matrix of initial connections (all connections between
495 % output nodes and the nodes listed in initialConnectedInputNodes). In
496 % nConnections, we add 1 since the bias is connected. vectorConnectionFrom is a
497 % row vector (i.e. one row, multiple columns) consists of a nOutputNodes times
498 % repeated row vector containing the ID of the initial connected input plus the
499 % bias node ID. For example, if there are 3 inputs (all connected) and 3
500 % outputs, the vector will
501 % [1,2,3,4],[1,2,3,4],[1,2,3,4]
502 % where 4 is the bias' node ID.
503 nConnections = (length(initialConnectedInputNodes)+1)*nOutputNodes;
504 vectorConnectionFrom = repmat([initialConnectedInputNodes,nInputNodes+1],[1,nOutputNodes]);
505 % vectorConnectionTo contains the node ID to which are connected every node
506 % listed in vectorConnectionFrom. For instance, if we take the example above
507 % with 3 inputs (all connected) and 3 outputs, we'd have:
508 % vectorConnectionT = [[5,5,5],[6,6,6],[7,7,7,7]]
509 vectorConnectionTo = [];
510 % Outputs' node IDs begin at the ID after the bias' ID which is why we take
511 % nInputNodes + 2. For instance, if we have 3 inputs + 1 bias, the IDs are 1,2,3,4
512 % are taken, so we start at 3 + 2 = 5 for the first output ID.
513 iOutputNodeStart = nInputNodes + 2;
514 iOutputNodeEnd = nInputNodes + 1 + nOutputNodes;
515 for iOutputNode = iOutputNodeStart:iOutputNodeEnd
516 % There is a +1 because the bias is connected.
517 vectorConnectionTo = ...
518 [vectorConnectionTo, ...
519 iOutputNode+iOutputNodes];
520 end
521 % Connection matrix is a matrix of two rows by (number of connections) columns
522 connectionMatrix = [...];
523 vectorConnectionFrom;
524 vectorConnectionTo;
525
526 % We build the initial population nodeGenes and codeGenes
527 for iIndividual = 1:populationSize
528 Population{iIndividual}.nodeGenes = [...;
529 1:(nInputNodes+nOutputNodes);
530 ones(1,nInputNodes),4,2*ones(1,nOutputNodes);
531 zeros(1,nInputNodes),1,zeros(1,nOutputNodes);
532 zeros(1,nInputNodes+nOutputNodes)];
533 % All weights uniformly distributed in [-1,+1], all connections enabled
534 Population{iIndividual}.connectionGenes = [...;
535 1:connections;
536 connectionMatrix;
537 rand(1,connections)-2/1;
538 ones(1,connections)];
539 Population{iIndividual}.fitness = 0;
540 Population{iIndividual}.species = 0;
541 end
542 innovationRecord = [...;
543 Population(populationSize).connectionGenes(1:3,:);
544 zeros(size(Population(populationSize).connectionGenes(1:2,:)));
545 % Highest node ID for initial population
546 innovationRecord(4,size(innovationRecord,2)) = max(Population(1).nodeGenes(1,:));
547
548 end
549
550 %% initialize_population_custom_gene()
551 function [Population,innovationRecord] = initialize_population_custom_gene...
552 populationSize...
553 startNodeGenes...
554 startConnectionGenes
555
556 % See initialize_population() for documentation
557
558 % Number of connections
559 nConnections = numel(startConnectionGenes(1,:));
560
561 % We build the initial population nodeGenes and codeGenes

```

```

562 for iIndividual = 1:populationSize
563 Population(iIndividual).nodeGenes = startNodeGenes;
564 % All weights uniformly distributed in [-1,+1]
565 Population(iIndividual).connectionGenes = [...];
566 startConnectionGenes(1:3,:);
567 rand(1,nConnections)*2-1;
568 startConnectionGenes(5,:);
569 Population(iIndividual).fitness = 0;
570 Population(iIndividual).species = 0;
571 end
572 innovationRecord = [...];
573 Population(populationSize).connectionGenes(1:3,:);
574 zeros(size(Population(populationSize).connectionGenes(1:2,:)));
575 % Highest node ID for initial population
576 innovationRecord(4,size(innovationRecord,2)) = max(Population(1).nodeGenes(1,:));
577
578 end
579
580 %% initial_specciation()
581 function [Population, SpeciesRecord] = ...
582     initial_specciation(initialConnectedInputNodes, Population, nOutputNodes, ...
583     SpeciesRecord, Speciation, loadGene)
584
585 % This function performs initial specciation. That is, it assigns each
586 % individual in a species according to their compatibility distance.
587
588 % Number of connections
589 if loadGene == 0
590     nConnections = (length(initialConnectedInputNodes)+1)*nOutputNodes;
591 else
592     nConnections = numel(Population(1).connectionGenes(1,:));
593 end
594 % Put first individual in species one and update SpeciesRecord
595 Population(1).species = 1;
596 % Species reference matrix (abbreviated, only weights, since there are
597 % no topology differences in initial population)
598 matrixReferenceIndividuals = Population(1).connectionGenes(4,:);
599 SpeciesRecord(1).id = 1;
600 SpeciesRecord(1).nIndividuals = 1;
601
602 % Loop through rest of individuals and either assign to existing species
603 % or create new species and use first individual of new species as
604 % reference
605 for iIndividual = 2:size(Population,2);
606     assignedExistingSpeciesFlag = 0;
607     newSpeciesFlag = 0;
608
609     % Loops through the existing species, terminates when either the
610     % i individual is assigned to existing species or there are no more
611     % species to test it against, which means it is a new species
612     while (assignedExistingSpeciesFlag == 0) && (newSpeciesFlag == 0)
613         % Computes compatibility distance, abbreviated, only average
614         % weight distance considered
615         averageWeightDiffMatchingGenes = ...
616             sum(...,
617                 abs(Population(iIndividual).connectionGenes(4,:) - ...
618                     matrixReferenceIndividuals(iSpecies,:)) ...
619             );
620         distance = Speciation.c3*averageWeightDiffMatchingGenes/nConnections;
621         % If within threshold, assign to the existing species
622         if distance < Speciation.threshold
623             Population(iIndividual).species = iSpecies;
624             assignedExistingSpeciesFlag = 1;
625             SpeciesRecord(iSpecies).nIndividuals = SpeciesRecord(iSpecies).nIndividuals+1;
626         end
627         iSpecies = iSpecies + 1;
628         % Outside of species references, must be new species
629         if (iSpecies > size(matrixReferenceIndividuals,1)) && (assignedExistingSpeciesFlag == 0)
630             newSpeciesFlag = 1;
631         end
632     end
633     % Check for new species, if it is, update the SpeciesRecord and
634     % use individual as reference for new species
635     if newSpeciesFlag == 1
636         Population(iIndividual).species = iSpecies;
637         matrixReferenceIndividuals = [...;
638             matrixReferenceIndividuals;
639             Population(iIndividual).connectionGenes(4,:)];
640         SpeciesRecord(iSpecies).id = iSpecies;
641         % If number individuals in a species is zero, that species is extinct
642         SpeciesRecord(iSpecies).nIndividuals = 1;
643     end
644 end
645
646 end
647
648 %% stagnation()
649 function [maxFitnessesCurrentGeneration, SpeciesRecord] = ...
650 stagnation(SpeciesRecord, Population, Stagnation, iGeneration)
651
652 % This function checks for stagnation. We also compute mean and max raw
653 % fitnesses in each species and store in SpeciesRecord.generationRecord
654
655 % Number of species in current generation
656 nSpecies = size(SpeciesRecord,2);
657
658 % Initialize maxFitnessesCurrentGeneration vector
659 maxFitnessesCurrentGeneration = zeros(1,nSpecies);
660
661 % The vector species contains the species index of each individual in the
662 % population
663 popSpecies = [Population().species];
664
665 % The vector fitness contains the fitness of each individual in the
666 % population
667 popFitness = [Population().fitness];
668
669 % Cycle through each specie
670 for iSpecies = 1:nSpecies
671     if SpeciesRecord(iSpecies).nIndividuals > 0
672         % Max fitness of current specie
673         [maxFitness, iMaxFitness] = max((popSpecies == iSpecies).*popFitness);
674         % Mean fitness of current specie
675         meanFitness = ...;
676         sum((popSpecies == iSpecies).*popFitness)/SpeciesRecord(iSpecies).nIndividuals;
677         % Number of generation in current specie
678         nGenerations = size(SpeciesRecord(iSpecies).generationRecord,2);
679         % Number of generations since last stagnation
680         % fitnesses plus current fitness
681         if nGenerations > Stagnation.nGenerations-2;
682             iGenerationStart = nGenerations - Stagnation.nGenerations + 2;
683             lastMaxFitnesses = ...
684                 SpeciesRecord(iSpecies).generationRecord(3,iGenerationStart:nGenerations);
685             stagnationVector = [lastMaxFitnesses,maxFitness];
686             % Average max fitnesses change for the last Stagnation.nGenerations generations
687             averageMaxFitnessesChange = abs(stagnationVector-mean(stagnationVector));
688             % Number of staged generation in current specie
689             nStagedGenerations = sum(averageMaxFitnessesChange < Stagnation.threshold);
690             % Check for stagnation
691             if nStagedGenerations == Stagnation.nGenerations
692                 % Set mean fitness to small value to eliminate species
693                 % (cannot be set to 0, if only one species is present,
694                 % we would have divide by zero in fitness sharing,
695                 % anyways, with only one species present, we have to keep it)
696                 meanFitness = 0.01;
697             end
698         end
699     % We add new generation to current specie
700     SpeciesRecord(iSpecies).generationRecord = [...;
701         SpeciesRecord(iSpecies).generationRecord,...;
702         [iGeneration;meanFitness;maxFitness;iMaxFitness]];
703     maxFitnessesCurrentGeneration(i,iSpecies) = maxFitness;
704
705     end
706 end
707 end
708
709 %% refocus()
710 function [SpeciesRecord] = ...
711     refocus(maxFitnessesCurrentGeneration, SpeciesRecord, Refocus)
712
713 % In rare cases when the fitness of the entire population does not
714 % improve for more than Refocus.nGenerations generations, only the top
715 % two species are allowed to reproduce, refocusing the search into the
716 % most promising spaces
717
718 % Index of the top species
719 iTopSpecies = max(maxFitnessesCurrentGeneration);
720
721 % Number of generations in the top specie
722 nGenerationsTopSpecie = size(SpeciesRecord(iTopSpecies).generationRecord,2);
723
724 % Check for refocus
725 if nGenerationsTopSpecie > Refocus.nGenerations
726     index1 = nGenerationsTopSpecie-Refocus.nGenerations;
727     index2 = nGenerationsTopSpecie;
728     % Max fitnesses from (nGenerationsTopSpecie-Refocus.nGenerations) to
729     % nGenerationsTopSpecie of top specie
730     maxFitnessesTopSpecie = SpeciesRecord(iTopSpecies).generationRecord(3,index1:index2);
731     % Average max fitnesses change for the top specie over the last
732     % nGenerationsTopSpecie generations
733     averageMaxFitnessesChangeTopSpecie = abs(maxFitnessesTopSpecie - mean(maxFitnessesTopSpecie));
734     % Number of staged generation in current specie
735     nStagedGenerationsTopSpecie = sum(averageMaxFitnessesChangeTopSpecie < Refocus.threshold);
736     % Check if number of staged generations in the top specie is equal
737     % to the threshold refocus generations number
738     if nStagedGenerationsTopSpecie == Refocus.nGenerations
739         % Sort max species' fitnesses from high fitnesses to low fitnesses
740         [-, vectorCull] = sort(-maxFitnessesCurrentGeneration);
741         % Index of the species to discard
742         vectorCull = vectorCull(1,:sum(maxFitnessesCurrentGeneration > 0));
743         for iSpecies = 1:size(vectorCull,2)
744             % Index of specie to discard
745             iCull = vectorCull(1,iSpecies);
746             % Index of the the last generation of the current specie to
747             % discard
748             iGenerationCull = size(SpeciesRecord(iCull).generationRecord,2);
749             % We assign a mean raw fitness of 0.01 to the last
750             % generation of the current specie to discard
751             SpeciesRecord(iCull).generationRecord(2,iGenerationCull) = 0.01;
752         end
753     end
754 end
755 end
756 end
757
758 %% reproduce()
759 function [NewPopulation, updatedSpeciesRecord, updatedInnovationRecord] = ...
760     reproduce(Population, SpeciesRecord, innovationRecord, Initial, Selection, ...
761     Crossover, Mutation, Speciation, iGeneration, populationSize)
762
763 %% Initial selection
764
765 %% Initial selection
766 [matExistingAndPropagSpecies, NewPopulation, Population, iIndividual] = ...
767     initial_selection(SpeciesRecord, Initial, Population, populationSize);
768
769 %% Generate reference population
770 populationRef = initialize_ref_population(Population, SpeciesRecord);
771
772 fprintf('\nExisting and propagating matrix:\n');
773 tmpMatExistingAndPropagSpecies = num2str(matExistingAndPropagSpecies, '%.4f');
774 fprintf(['tmpMatExistingAndPropagSpecies(1,:); \n']);
775 fprintf(['tmpMatExistingAndPropagSpecies(2,:); \n']);
776 fprintf(['tmpMatExistingAndPropagSpecies(3,:); \n\n']);
777
778 %% Reproduction
779
780 %% Cycle through all existing species
781 for iSpecies = 1:size(matExistingAndPropagSpecies,2)
782
783     % Number of individuals in specie
784     countIndividualsSpecies = 0;
785
786     % This is the ID of species which will be reproduced this cycle.
787     % IMPORTANT: iSpecies only has relevance to
788     % matExistingAndPropagSpecies, all other mechanisms using
789     % species in some way must use specieId
790     specieId = matExistingAndPropagSpecies(1,iSpecies);
791
792     % Linear Ranking and stochastic universal sampling Ranking with
793     % Selection.pressure
794     [NewChrIx, numberCrossover, numberMutate] = ...
795         ranking_and_sampling(Population, Selection, specieId, iSpecies, ...
796             matExistingAndPropagSpecies, Crossover);
797
798     % Cycle until actual number of offspring has reached allotted number of offspring
799     while matExistingAndPropagSpecies(3,iSpecies) < matExistingAndPropagSpecies(2,iSpecies)
800
801         % Increment number of individuals in population
802         iIndividual = iIndividual + 1;
803
804         % Increment number of individuals in current specie
805         countIndividualsSpecies = countIndividualsSpecies + 1;
806
807         % Crossover
808         if countIndividualsSpecies <= numberCrossover
809             % OK we are doing crossover
810             NewIndividual = crossover(...,
811                 Population, NewChrIx, countIndividualsSpecies, Crossover, ...
812                 matExistingAndPropagSpecies);
813         else
814             % No crossover, just copy a individual of the species and mutate in
815             % subsequent steps
816             NewIndividual = Population(NewChrIx(numberCrossover+countIndividualsSpecies));
817         end
818
819         % Hidden nodes culling
820         NewIndividual = node_culling(NewIndividual);
821
822         % Disabled Genes Mutation
823         NewIndividual = enable_gene_mutation(NewIndividual, Mutation);
824
825         % Weight Mutation
826         NewIndividual = weight_mutation(NewIndividual, Mutation);
827
828         % IMPORTANT: The checks for duplicate innovations in the following
829         % two types of mutation can only check in the current generation
830
831         flag1 = rand() < Mutation.probabilityAddNode;
832         if flag1 == 0
833             % Add Connection Mutation
834             [NewIndividual, innovationRecord] = ...
835                 add_connection_mutation(NewIndividual, innovationRecord, ...
836                 Mutation, iGeneration, flag1);
837         elseif flag1 == 1
838             % Add (insert) Node Mutation
839             [NewIndividual, innovationRecord] = ...
840                 add_node_mutation(innovationRecord, iGeneration, NewIndividual);
841         end
842
843         % Speciation
844         [NewIndividual, SpeciesRecord, populationRef] = ...
845             speciation(NewIndividual, SpeciesRecord, populationRef, Speciation);

```

```

846
847 % Add NewIndividual to NewPopulation
848 NewPopulation(iIndividual) = NewIndividual;
849
850 % Increment species
851 matExistingAndPropagSpecies(3,iSpecies) = ...
852     matExistingAndPropagSpecies(3,iSpecies) + 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 iSpecies = 1:size(SpeciesRecord,2)
860     SpeciesRecord(iSpecies).nIndividuals = sum([NewPopulation(:).species] == iSpecies);
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 fitnesses
875 sumAverageFitnesses = 0;
876 for iSpecies = 1:size(SpeciesRecord,2)
877     % Number of generation in current specie
878     nGenerations = size(SpeciesRecord(iSpecies).generationRecord,2);
879     % Mean raw fitness of the last generation of current specie "iSpecies"
880     meanRawFitness = SpeciesRecord(iSpecies).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(iSpecies).nIndividuals > 0);
885 end
886
887 end
888
889 %% initial_selection()
890 function [matExistingAndPropagSpecies, NewPopulation, Population, iIndividual] = ...
891     initial_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 allotted 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 % (only one specie is not dying out, i.e. has at least one individual
910 % allotted to it) 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 fitnesses
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 iSpecies = 1:size(SpeciesRecord,2)
933     % Test if species existed in old generation
934     if SpeciesRecord(iSpecies).nIndividuals > 0
935
936         % Number of generation in current specie
937         nGenerations = size(SpeciesRecord(iSpecies).generationRecord,2);
938         % Mean raw fitness of current generation
939         meanRawFitness = SpeciesRecord(iSpecies).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         % since it's floored
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(iSpecies).id];
962             nOffsprings;
963             0];
964
965         % Check for condition for objective 2
966         if SpeciesRecord(iSpecies).nIndividuals >= Initial.numberCopy
967             iIndividual = iIndividual + 1;
968             % Objective 2
969             NewPopulation(iIndividual) = ...
970                 Population(SpeciesRecord(iSpecies).generationRecord(4,nGenerations));
971             % Update matExistingAndPropagSpecies
972             matExistingAndPropagSpecies(3,size(matExistingAndPropagSpecies,2)) = 1;
973         end
974
975 % condition1 checks if specie has more individuals than numberForKill
976 condition1 = SpeciesRecord(iSpecies).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(iSpecies).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 iSpecies
987     iIndividualsToKill = find(popSpecies == iSpecies);
988
989 % Matrix containing on first row the index of individual in
990 % current specie to kill and their fitness on the second row
991 matrixIndividualsSpecies = [...];
992     iIndividualsToKill;
993     [Population(iIndividualsToKill).fitness];
994 % We sort the individual in the specie according to their fitness
995     [, sortingVector] = sort(matrixIndividualsSpecies(2,:));
996 % Sorted matrixIndividualsSpecies
997 matrixIndividualsSpecies = matrixIndividualsSpecies(:,sortingVector);
998 % Index of sorted individual in specie
999 sortingVector = matrixIndividualsSpecies(1,:);
1000 % MATLAB actually does not offer a facility for redirecting the
1001 % pointers which link one element in a structure with the next,
1002 % so essentially this individual entry in the population
1003 % structure cannot be erased. Rather, it will have its species'
1004 % ID set to zero, which has the same effect of removing it from
1005 % the rest of the reproduction cycle, since all reproduction
1006 % functions access the population structure through the
1007 % species' ID and no species has an ID of zero.
1008 % We kill the Initial.killPercentage first individual in the specie
1009 nKills = floor(SpeciesRecord(iSpecies).nIndividuals*Initial.killPercentage);
1010 for ikill = 1:nKills
1011     % Objective 3
1012     Population(sortingVector(ikill)).species = 0;
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 iSpeciesRef = 1:nSpecies
1037     % Index of individuals in current specie (iSpeciesRef) that exists in
1038     % old population. By exist, we mean the individuals that haven't been
1039     % killed during initial_selection()
1040     iExistingIndividual = popSpecies == iSpeciesRef;
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, iSpecies, ...
1061     matExistingAndPropagSpecies, Crossover)
1062
1063 % This function compute the selected 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/doc/alгинdex-02.html for explanation
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 specieId
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     else
1087         FitnV = FitnV';
1088     end
1089
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,iSpecies) - ...
1094     matExistingAndPropagSpecies(3,iSpecies);
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 Algorithms 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 vector containing the indexes of the selected
1123 % individuals relative to the original population, shuf
1124 NewChrIx = NewChrIx(shuf);
1125 % Relate to indexes of individuals in population
1126 NewChrIx = iFitnessesSpecies(NewChrIx);
1127
1128 end
1129

```

```

1130 % crossover()
1131 function NewIndividual = crossover...
1132 Population, NewChrIx, countIndividualsSpecies, Crossover, ...
1133 matExistingAndPropagSpecies)
1134
1135 % Select Parent1
1136 Parent1 = Population(NewChrIx(2*countIndividualsSpecies-1));
1137
1138 % Select Parent2
1139 foundParent2 = 0;
1140 % We select from other species (can only be done if there
1141 % is more than one species in old population)
1142 condition1 = rand() < Crossover.probabilityInterspecies;
1143 condition2 = size(matExistingAndPropagSpecies,2) > 1;
1144 if condition1 && condition2
1145 while foundParent2 == 0
1146 [_, iParent2] = max(rand(1,size(Population,2)));
1147 Parent2 = Population(iParent2);
1148 % Check if Parent2.species is not species 0 (deleted
1149 % individual) or species of Parent1
1150 foundParent2 = ...
1151 ((Parent2.species == 0) & (Parent2.species == Parent1.species));
1152 end
1153 % Set fitnesses to same to ensure that disjoint and excess
1154 % genes are inherited fully from both parents (tip from ken)
1155 Parent2.fitness = Parent1.fitness;
1156
1157 else
1158 % OK we take Parent2 from same species as Parent1
1159 Parent2 = Population(NewChrIx(2*countIndividualsSpecies));
1160 end
1161
1162 % Inherit nodes from both parents
1163 NewIndividual.nodeGenes = [];
1164 nGenesP1 = size(Parent1.nodeGenes,2);
1165 nGenesP2 = size(Parent2.nodeGenes,2);
1166 matrixNodeLineup = [...;
1167 [Parent1.nodeGenes(1,:);zeros(1,nGenesP1)],...
1168 [Parent2.nodeGenes(1,:);zeros(1,nGenesP2)];1:nGenesP2];
1169 [..., sortNodeVec] = sort(matrixNodeLineup(:,1));
1170 matrixNodeLineup = matrixNodeLineup(:,sortNodeVec);
1171 nodeNumber = 0;
1172 for iNodeSort = 1:size(matrixNodeLineup,2)
1173 if nodeNumber ~= matrixNodeLineup(1,iNodeSort)
1174 if matrixNodeLineup(2,iNodeSort) > 0
1175 NewIndividual.nodeGenes = ...
1176 [NewIndividual.nodeGenes, ...
1177 Parent1.nodeGenes(:,matrixNodeLineup(2,iNodeSort))];
1178 else
1179 NewIndividual.nodeGenes = ...
1180 [NewIndividual.nodeGenes, ...
1181 Parent2.nodeGenes(:,matrixNodeLineup(3,iNodeSort))];
1182 end
1183 nodeNumber = matrixNodeLineup(1,iNodeSort);
1184 end
1185
1186 % Crossover connection genes
1187 % First do lineup of connection genes
1188 nConnGenesP1 = size(Parent1.connectionGenes,2);
1189 nConnGenesP2 = size(Parent2.connectionGenes,2);
1190 matrixLineup = [...;
1191 [Parent1.connectionGenes(1,:);1:nConnGenesP1];zeros(1,nConnGenesP1)],...
1192 [Parent2.connectionGenes(1,:);zeros(1,nConnGenesP2);1:nConnGenesP2]];
1193 [..., sortNodeVec] = sort(matrixLineup(:,1));
1194 matrixLineup = matrixLineup(:,sortNodeVec);
1195 finalMatrixLineup = [];
1196 innovationNumber = 0;
1197 for iSort = 1:size(matrixLineup,2)
1198 if innovationNumber == matrixLineup(1,iSort)
1199 finalMatrixLineup = [finalMatrixLineup, matrixLineup(:,iSort)];
1200 innovationNumber = matrixLineup(1,iSort);
1201 else
1202 finalMatrixLineup(2:3, size(finalMatrixLineup,2)) = ...
1203 finalMatrixLineup(2:3, size(finalMatrixLineup,2)) + ...
1204 matrixLineup(2:3,iSort);
1205 end
1206 end
1207
1208 % OK Connection Genes are lined up, start with crossover
1209 NewIndividual.connectionGenes = [];
1210 for iLineup = 1:size(finalMatrixLineup,2)
1211 % Check for matching genes do crossover
1212 if (finalMatrixLineup(2,iLineup) > 0) && (finalMatrixLineup(3,iLineup) > 0)
1213 % Random crossover for matching genes
1214 if rand() < 0.5
1215 NewIndividual.connectionGenes = ...
1216 [NewIndividual.connectionGenes, ...
1217 Parent1.connectionGenes(:,finalMatrixLineup(2,iLineup))];
1218 else
1219 NewIndividual.connectionGenes = ...
1220 [NewIndividual.connectionGenes, ...
1221 Parent2.connectionGenes(:,finalMatrixLineup(3,iLineup))];
1222 end
1223 % Weight averaging for offspring, otherwise the randomly inherited
1224 % weights are left undisturbed
1225 if rand() < Crossover.probabilityMultipoint
1226 NewIndividual.connectionGenes(4, size(NewIndividual.connectionGenes,2)) = ...
1227 (...,
1228 Parent1.connectionGenes(4,finalMatrixLineup(2,iLineup)) + ...
1229 Parent2.connectionGenes(4,finalMatrixLineup(3,iLineup))...
1230 );/2;
1231 end
1232 end
1233 % Test if there exist further connection genes from
1234 % iLineup+1 to end of finalMatrixLineup for Parent1 (to
1235 % detect excess)
1236 parent1Flag = sum(finalMatrixLineup(2,iLineup+1:size(finalMatrixLineup,2)));
1237 % Test if there exist further connection genes from
1238 % iLineup+1 to end of finalMatrixLineup for Parent1 (to
1239 % detect excess)
1240 parent2Flag = sum(finalMatrixLineup(3,iLineup+1:size(finalMatrixLineup,2)));
1241 % Two cases to check (excess is taken care of in the disjoint gene checks)
1242 % Disjoint Parent1
1243 if (finalMatrixLineup(2,iLineup) > 0) && (finalMatrixLineup(3,iLineup) == 0)
1244 if Parent1.fitness > Parent2.fitness
1245 NewIndividual.connectionGenes = ...
1246 [NewIndividual.connectionGenes, ...
1247 Parent1.connectionGenes(:,finalMatrixLineup(2,iLineup))];
1248 end
1249 end
1250 % Disjoint Parent2
1251 if (finalMatrixLineup(2,iLineup) == 0) && (finalMatrixLineup(3,iLineup) > 0)
1252 if Parent2.fitness > Parent1.fitness
1253 NewIndividual.connectionGenes = ...
1254 [NewIndividual.connectionGenes, ...
1255 Parent2.connectionGenes(:,finalMatrixLineup(3,iLineup))];
1256 end
1257 end
1258 end
1259
1260 % Has no impact on algorithm, only required for assignment to new population
1261 NewIndividual.fitness = 0;
1262
1263 % Will be species hint for speciation
1264 NewIndividual.species = Parent1.species;
1265
1266 end
1267
1268 % node_culling()
1269 function NewIndividual = node_culling(NewIndividual)
1270
1271 % Hidden nodes culling (remove any hidden nodes where there is no
1272 % corresponding connection gene in the new individual)
1273 connectedNodes = [];
1274 for iNodeCulling = 1:size(NewIndividual.nodeGenes,2)
1275 nodeConnectedFlag = ...
1276 sum(NewIndividual.connectionGenes(2,:) == NewIndividual.nodeGenes(1,iNodeCulling)) + ...
1277 sum(NewIndividual.connectionGenes(3,:) == NewIndividual.nodeGenes(1,iNodeCulling));
1278 if (nodeConnectedFlag > 0) || (NewIndividual.nodeGenes(2,iNodeCulling) == 3)
1279 connectedNodes = [connectedNodes, NewIndividual.nodeGenes(:,iNodeCulling)];
1280 end
1281 end
1282 NewIndividual.nodeGenes = connectedNodes;
1283
1284 end
1285
1286 % enable_gene_mutation()
1287 function NewIndividual = enable_gene_mutation(NewIndividual, Mutation)
1288
1289 % Disabled Genes Mutation
1290 % Run through all connection genes in a NewIndividual, find disabled
1291 % connection genes, enable again with Crossover.probabilityGeneReenabled
1292 % probability
1293 for iConnectionGene = 1:size(NewIndividual.connectionGenes,2)
1294 condition1 = NewIndividual.connectionGenes(5,iConnectionGene) == 0;
1295 condition2 = rand() < Mutation.probabilityGeneReenabled;
1296 if condition1 && condition2
1297 NewIndividual.connectionGenes(5,iConnectionGene) = 1;
1298 end
1299 end
1300
1301 end
1302
1303 % weight_mutation()
1304 function NewIndividual = weight_mutation(NewIndividual, Mutation)
1305
1306 % Weight Mutation
1307 % Run through all connection genes in a NewIndividual, decide on
1308 % mutating or not
1309 for iConnectionGene = 1:size(NewIndividual.connectionGenes,2)
1310 % * iConnectionGene/size(NewIndividual.connectionGenes,2)
1311 % linearly biased towards higher probability of mutation at end of
1312 % connection genes
1313 if rand() < Mutation.probabilityMutateWeight
1314 NewIndividual.connectionGenes(4,iConnectionGene) = ...
1315 NewIndividual.connectionGenes(4,iConnectionGene) + ...
1316 Mutation.weightRange*(rand() - 0.5);
1317 end
1318 % Weight of connection genes of NewIndividual
1319 weightConnGenes = NewIndividual.connectionGenes(4,iConnectionGene);
1320 % Weight capping
1321 NewIndividual.connectionGenes(4,iConnectionGene) = ...
1322 weightConnGenes*(abs(weightConnGenes) <= Mutation.weightCap) + ...
1323 (sign(weightConnGenes)*Mutation.weightCap)*(abs(weightConnGenes) > Mutation.weightCap);
1324 end
1325
1326 end
1327
1328 % add_connection_mutation()
1329 function [NewIndividual, innovationRecord] = ...
1330 add_connection_mutation(NewIndividual, innovationRecord, Mutation, ...
1331 iGeneration, flag1)
1332
1333 % Add Connection Mutation
1334 flagRecurrencyEnabled = rand() < Mutation.probabilityRecurrency;
1335
1336 % Connections can run from every node
1337 vectorPossibleConnectFromNodes = NewIndividual.nodeGenes(1,:);
1338
1339 % Connections can only run into hidden and output nodes
1340 vectorPossibleConnectToNodes = ...
1341 NewIndividual.nodeGenes(...,1);
1342 i,find((NewIndividual.nodeGenes(2,:)== 2) + (NewIndividual.nodeGenes(2,:)== 3))...
1343 );
1344 numberPossibleConnection = ...
1345 length(vectorPossibleConnectFromNodes)*length(vectorPossibleConnectToNodes) - ...
1346 size(NewIndividual.connectionGenes,2);
1347
1348 % Check if new connections can be added to genes (if there are any
1349 % possible connections which are not already existing in genes of
1350 % new individual)
1351 condition1 = rand() < Mutation.probabilityAddConnection;
1352 condition2 = numberPossibleConnection > 0;
1353 condition3 = flag1 == 0;
1354 if condition1 && condition2 && condition3
1355 % First build matrix containing all possible new connection for
1356 % nodegene of new individual
1357 newConnectionMatrix = [];
1358 for iConnectFrom = 1:length(vectorPossibleConnectFromNodes)
1359 for iConnectTo = 1:length(vectorPossibleConnectToNodes)
1360 possibleConnection = ...
1361 [...,
1362 vectorPossibleConnectFromNodes(iConnectFrom);
1363 vectorPossibleConnectToNodes(iConnectTo));
1364 % Check if proposed connection is not already contained in gene
1365 condition4 = sum((...
1366 (NewIndividual.connectionGenes(2,:) == possibleConnection(1)).* ...
1367 (NewIndividual.connectionGenes(3,:) == possibleConnection(2)) ...
1368 )) == 0;
1369 if condition4
1370 newConnectionMatrix = [newConnectionMatrix, possibleConnection];
1371 end
1372 end
1373 end
1374 % Shuffle possible new connections randomly
1375 [..., shuffle] = sort(rand(1,size(newConnectionMatrix,2)));
1376 newConnectionMatrix = newConnectionMatrix(:,shuffle);
1377
1378 iNewConnection = 0;
1379 flagConnection0 = 0;
1380 % Check if connection is ok. (meaning either non-recurrent or
1381 % recurrent and flagRecurrencyEnabled set to 1) if no
1382 % connection is found which is ok, no connection will be added
1383 % to connection genes of new individual
1384 while (flagConnection0 == 0) && (iNewConnection < size(newConnectionMatrix,2))
1385 iNewConnection = iNewConnection + 1;
1386 newConnection = newConnectionMatrix(:,iNewConnection);
1387 % Test new connection if it is recurrent (i.e. at least one
1388 % of the possible path starting from connect_to node in
1389 % the new connection ends back to the connect_from node
1390 flagRecurrent = 0;
1391 % Trivial recurrency
1392 if newConnection(1) == newConnection(2)
1393 flagRecurrent = 1;
1394 end
1395 nodesCurrentLevel = newConnection(2);
1396 depth = 0;
1397 while (flagRecurrent == 0) && ...
1398 (depth < size(NewIndividual.connectionGenes,2)) && ...
1399 ~isempty(nodesCurrentLevel)
1400 depth = depth + 1;
1401 nodesNextLevel = [];
1402 for indexCheck = 1:size(nodesCurrentLevel),
1403 indexTemp = find((...
1404 NewIndividual.connectionGenes(2,:) == ...
1405 nodesCurrentLevel(indexCheck)) ...
1406 ));
1407 nodesNextLevel = ...
1408 [nodesNextLevel, ...
1409 NewIndividual.connectionGenes(3,indexTemp)];
1410 end
1411 if sum(nodesNextLevel) == newConnection(1) > 0
1412 flagRecurrent = 1;
1413 end

```

```

1414     nodesCurrentLevel = nodesNextLevel;
1415   end
1416   if flagRecurrent == 0
1417     flagConnectionOk = 1;
1418   elseif flagRecurrencyEnabled
1419     flagConnectionOk = 1;
1420   end
1421 end
1422
1423 % Now we test if it is a true innovation (i.e. hasn't already
1424 % happened in this generation) we can only do this if a valid
1425 % new connection has been found
1426 if flagConnectionOk
1427   % Set flag signifying new innovation (connection not
1428 % contained in innovationRecord of this generation)
1429   iAlreadyHappened = find(
1430     (innovationRecord(5,:)) == iGeneration).* ...
1431     (innovationRecord(2,:)) == newConnection(1).* ...
1432     (innovationRecord(3,:)) == newConnection(2) ...
1433   );
1434
1435 newInnovation = not(sum(iAlreadyHappened));
1436 if newInnovation == 1 % OK is new innovation
1437   % Update the new connection with its innovation number
1438   newConnection = [max(innovationRecord(1,:));newConnection];
1439   % Update connection.genes. Weight is uniformly distributed between
1440   % [-1,+1]
1441   NewIndividual.connectionGenes = [...;
1442     NewIndividual.connectionGenes.* ...
1443     (rand(1,NewIndividual.sizeInnovationRecord)*2-1)];
1444   % Update innovationRecord
1445   innovationRecord = [innovationRecord,newConnection;0;iGeneration];
1446 else % Connection gene already exists in innovationRecord of this generation
1447   % Update connectionGenes. Weight is uniformly distributed between
1448   % [-1,+1]
1449   NewIndividual.connectionGenes = [...;
1450     NewIndividual.connectionGenes.* ...
1451     (innovationRecord(1:3,iAlreadyHappened);rand()*2-1)];
1452 end
1453
1454 end
1455
1456 %% add_node_mutation()
1457 function [NewIndividual, innovationRecord] = ...
1458   add_node_mutation(innovationRecord, iGeneration, NewIndividual)
1459
1460 % Add (Insert) Node Mutation
1461 newInnovation = 0;
1462 % Highest innovation number from last generation (to ensure
1463 % that only connections from from last generation or older are
1464 % chosen for add node mutation, otherwise a new connection
1465 % added in the last mutation might instantly be disabled)
1466 maxOldInnovationNumber = max(...;
1467   (innovationRecord(5,:)) < iGeneration).*innovationRecord(1,:);
1468 );
1469
1470 % Compute vector of connections into which a new node could be
1471 % inserted and their positions in the connectionGene matrix.
1472 % This vector is composed of all nondisabled connections which
1473 % stem at least from the last generation or older
1474 indexTemp = find(...;
1475   (NewIndividual.connectionGenes(5,:)) == 1) & ...
1476   (NewIndividual.connectionGenes(1,:)) <= maxOldInnovationNumber) ...
1477 );
1478
1479 vectorPossibleConnections = [...;
1480   NewIndividual.connectionGenes(2:3,indexTemp);
1481   indexTemp];
1482
1483 insertNodeConnection = ...
1484   vectorPossibleConnections(:,round(rand()*size(vectorPossibleConnections,2))+0.5);
1485 % Set provisionally to 1, will be checked
1486 newInnovation = 1;
1487
1488 % Beginning of check innovation record to test for real
1489 % innovation. existInnovation contains vector of index of
1490 % elements in innovation record which fulfill three things:
1491 % current generation, add node mutation and same connect from
1492 % as current innovation
1493 existInnovation = find((innovationRecord(5,:)) == ...
1494   iGeneration).* ...
1495   (innovationRecord(4,:)) > 0).*...
1496   (innovationRecord(2,:)) == insertNodeConnection(1));
1497
1498 % If this is the first, we have to test for connect_to node
1499 % to see if innovation really is the same
1500 if sum(existInnovation) == 0
1501   for indexCheck = 1:length(existInnovation)
1502     if innovationRecord(3,existInnovation(indexCheck)+1) == insertNodeConnection(2)
1503       newInnovation = 0;
1504       iAlreadyExistentThisGeneration = existInnovation(indexCheck);
1505     end
1506   end
1507
1508 if newInnovation == 1 % OK is true innovation for current generation
1509 % Update node genes. By [newNodeNumber;3;0;0], we see that only hidden nodes
1510 % can be added by add_node_mutation.
1511 newNodeNumber = max(innovationRecord(4,:)) + 1;
1512 NewIndividual.nodeGenes = [NewIndividual.nodeGenes,[newNodeNumber;3;0;0]];
1513 % Disable old connection gene
1514 NewIndividual.connectionGenes(5,insertNodeConnection(3)) = 0;
1515 % Update connectionGenes. newConnections is a 5 rows by two columns matrix
1516 % where the first column is the new connection TO the new node whereas
1517 % the second column is the connection FROM the new node
1518 newConnections = [...;
1519   [max(innovationRecord(1,:)) + 1;
1520    insertNodeConnection(1);
1521    newNodeNumber;
1522    1;
1523    1];
1524   [max(innovationRecord(1,:)) + 2;
1525    insertNodeConnection(2);
1526    NewIndividual.connectionGenes(4,insertNodeConnection(3));
1527    1];
1528 ];
1529
1530 % Extend connection genes by the two new connections
1531 NewIndividual.connectionGenes = [NewIndividual.connectionGenes, newConnections];
1532 % Update innovationRecord
1533 innovationRecord = [...;
1534   innovationRecord(1:3,:);newNodeNumber,0;iGeneration,iGeneration];
1535
1536 else % No new innovation, has already happened at least once in this generation
1537   % Update node genes
1538   nodeNumber = innovationRecord(4,iAlreadyExistentThisGeneration);
1539   NewIndividual.nodeGenes = [NewIndividual.nodeGenes,[nodeNumber;3;0;0]];
1540   % Update connectionGenes
1541   % Disable old connection gene
1542   NewIndividual.connectionGenes(5,insertNodeConnection(3)) = 0;
1543   newConnections = [...;
1544     innovationRecord(1:3,iAlreadyExistentThisGeneration:iAlreadyExistentThisGeneration+1);
1545     NewIndividual.connectionGenes(4,insertNodeConnection(3));1,1];
1546   % Length of the connection genes of current NewIndividual
1547   lengthConGen = size(NewIndividual.connectionGenes,2);
1548   % Check if there was an add_connection_mutation to current
1549   % NewIndividual which has a higher innovation number than
1550   % current add_node_mutation
1551   if NewIndividual.connectionGenes(1,lengthConGen) > newConnections(1,2)
1552     NewIndividual.connectionGenes = [...;
1553       NewIndividual.connectionGenes(1:lengthConGen-1), ...
1554       newConnections, ...
1555       NewIndividual.connectionGenes(:,lengthConGen)];
1556   else
1557     NewIndividual.connectionGenes = [...;
1558       NewIndividual.connectionGenes, ...
1559       newConnections];
1560
1561   end
1562 end
1563
1564 %% speciation()
1565 function [NewIndividual, SpeciesRecord, populationRef, Speciation] = ...
1566   speciation(NewIndividual, SpeciesRecord, populationRef, Speciation)
1567
1568 % This function performs speciation to assign the new individual into an
1569 % existing or new specie
1570
1571 % Loop through comparison vector
1572 speciesAssigned = 0;
1573 iPopulationRef = 0;
1574 while (speciesAssigned == 0) && (iPopulationRef < size(populationRef,2))
1575   % Extract referenceIndividual from reference population
1576   iPopulationRef = iPopulationRef + 1;
1577   referenceIndividual = populationRef(iPopulationRef);
1578   % Run through both connection genes, compute disjoint, excess,
1579   % and average weight difference.
1580   % Maximum number of genes between new individual and reference individual
1581   maxNumGenes = max(...;
1582     [size(NewIndividual.connectionGenes,2),...
1583      size(referenceIndividual.connectionGenes,2)]);
1584
1585 % Maximum innovation number between new individual and reference individual
1586 maxNumInnovation = max(...;
1587   [NewIndividual.connectionGenes(1,:),...
1588     referenceIndividual.connectionGenes(1,:)]);
1589
1590 % New innovation vector
1591 vectorInnovationNew = [zeros(1,max(NewIndividual.connectionGenes(1,:))),...
1592   ones(1,maxNumInnovation - max(NewIndividual.connectionGenes(1,:)))];
1593
1594 % New weight vector
1595 vectorWeightNew = zeros(1,maxNumInnovation);
1596 vectorWeightNew(NewIndividual.connectionGenes(1,:)) = ...
1597   NewIndividual.connectionGenes(4,:);
1598
1599 % Innovation vector for reference individual
1600 vectorInnovationRef = [...;
1601   4*ones(1,max(referenceIndividual.connectionGenes(1,:))),...
1602   8*ones(1,maxNumInnovation - max(referenceIndividual.connectionGenes(1,:)))];
1603
1604 % Weight vector for reference individual
1605 vectorWeightRef = [vectorInnovationRef(1,:)];
1606 vectorWeightRef(referenceIndividual.connectionGenes(1,:)) = ...
1607   referenceIndividual.connectionGenes(4,:);
1608
1609 % Lineup vector between new individual and reference individual
1610 vectorLineup = vectorInnovationNew + vectorInnovationRef;
1611
1612 % Number of excess genes between new and reference individuals
1613 excess = sum(vectorLineup == 10) + sum(vectorLineup == 17);
1614
1615 % Number of disjoint genes between new and reference individuals
1616 disjoint = sum(vectorLineup == 6) + sum(vectorLineup == 16);
1617
1618 % Matching genes
1619 vectorMatching = find(vectorLineup == 18);
1620
1621 % Average weight difference
1622 averageWeightDifference = ...
1623   sum(...;
1624     abs(vectorWeightNew(vectorMatching) - vectorWeightRef(vectorMatching)));
1625
1626 % Compute compatibility distance between new and reference individuals
1627 maxNumGenes = Speciation.c1*excess/maxNumGenes + ...
1628   Speciation.c2*disjoint/maxNumGenes + ...
1629   Speciation.c3*averageWeightDifference;
1630
1631 if distance < Speciation.threshold
1632   % Assign individual to same species as current reference individual
1633   NewIndividual.species = referenceIndividual.species;
1634   % Set flag indicating NewIndividual has been assigned to species
1635   speciesAssigned = 1;
1636 end
1637
1638 end
1639
1640 % Not compatible with any? well, then create new species
1641 if speciesAssigned == 0
1642   newSpeciesId = size(SpeciesRecord,2) + 1;
1643   % Assign individual to new species
1644   NewIndividual.species = newSpeciesId;
1645   % Update SpeciesRecord
1646   SpeciesRecord(newSpeciesId).id = newSpeciesId;
1647   SpeciesRecord(newSpeciesId).nIndividuals = 1;
1648   SpeciesRecord(newSpeciesId).generationRecord = [];
1649   % Update population reference
1650   populationRef(size(populationRef,2)+1) = NewIndividual;
1651 end
1652
1653 end
1654
1655 end
1656
1657 %% compute_stats()
1658 function [nAverageNonDisabledConnections, nAverageHiddenNodes, maxFitness, ...
1659   maxOverallFitness, nAverageDisabledConnections, ...
1660   nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie] = ...
1661   compute_stats(Population, populationSize, iGeneration, ...
1662   SpeciesRecord, nAverageNonDisabledConnections, nAverageHiddenNodes, ...
1663   maxOverallFitness, nAverageDisabledConnections, ...
1664   nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie)
1665
1666 % Number of individuals
1667 nIndividuals = size(Population,2);
1668
1669 % Compute the average number of enabled connections and hidden nodes in current
1670 % population
1671 a = 0; % Sum of enabled connection in current population
1672 b = 0; % Sum of hidden nodes in current population
1673 c = 0; % Sum of disabled connection in current population
1674 for iIndividual = 1:nIndividuals
1675   a = a + sum(Population(iIndividual).connectionGenes(5,:)) == 1;
1676   b = b + sum(Population(iIndividual).nodeGenes(2,:)) == 3;
1677   c = c + sum(Population(iIndividual).connectionGenes(5,:)) == 0;
1678 end
1679
1680 nAverageNonDisabledConnections = [...;
1681   nAverageNonDisabledConnections, ...
1682   a/(populationSize*iGeneration)];
1683
1684 nAverageDisabledConnections = [...;
1685   nAverageDisabledConnections, ...
1686   b/(populationSize*iGeneration)];
1687
1688 nAverageHiddenNodes = [...;
1689   nAverageHiddenNodes, ...
1690   c/(populationSize*iGeneration)];
1691
1692 % Number of species in current generation
1693 nSpecies = size(SpeciesRecord,2);
1694 nSpeciesArray = [nSpeciesArray,nSpecies];
1695 nSpeciesArray = [nSpeciesArray,nuelnunique([Population().species])];
1696
1697 % Average number of individuals per specie
1698 uniqueSpecieId = unique([Population().species]);
1699 nIndPerSpecieTp = zeros(1,nuelnunique(uniqueSpecieId));
1700 for k = 1:nuelnunique(uniqueSpecieId)
1701   nIndPerSpecieTp(k) = sum([Population().species] == uniqueSpecieId(k));
1702 end
1703 nIndPerSpecie = [nIndPerSpecie,mean(nIndPerSpecieTp)];
1704
1705 % We build the fitCurGenSpecie matrix with 3 rows and nSpecies columns. The
1706 % first row is the index of the current generation of the specie. The second row
1707 % is mean raw fitness of the current generation for the specie and the third row
1708 % is the max raw fitness of the current generation for the specie.
1709 fitCurGenSpecie = [];
1710 for iSpecie = 1:nSpecies
1711   % Number of generations in current specie
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3087
3088
3089
3089
3090
3091
3092
3093
3093
3094
3095
3095
3096
3097
3098
3099
3099
3100
3101
3102
3103
3103
3104
3105
3106
3106
3107
3108
3109
3109
3110
3111
3112
3112
3113
3114
3114
3115
3116
3116
3117
3118
3118
3119
3119
3120
3121
3121
3122
3123
3123
3124
3125
3125
3126
3127
3127
3128
3128
3129
3129
3130
3
```

```

1698 nGenerations = size(SpeciesRecord(iSpecies).generationRecord,2);
1699 fitCurrGenSpecie = [...];
1700 fitCurrGenSpecie...
1701 SpeciesRecord(iSpecies).generationRecord(1:3,nGenerations)];
1702 end
1703
1704 % Maximal fitness for current generation
1705 maxFitness = max(fitCurrGenSpecie(3,:)*(fitCurrGenSpecie(1,:) == iGeneration));
1706 fprintf('Max fitness is %.6f\n',maxFitness);
1707
1708 % Maximum overall fitness. It is a matrix with 2 rows and nGenerations columns.
1709 % The first row is the maximum overall fitness of the current generation and the
1710 % second row is the generation number.
1711 maxOverallFitness = [...];
1712 maxOverallFitness...
1713 [maxFitness;iGeneration];
1714
1715 % Mean fitness for current generation
1716 meanFitness = mean([Population().fitness]);
1717 meanOverallFitness = [...];
1718 meanOverallFitness...
1719 [meanFitness;iGeneration];
1720
1721 % Number of recurrent connections
1722 nRecurrentConnections = ...
1723 find_recurrent_connection(Population);
1724 meanRecurrentConnection = [meanRecurrentConnection,mean(nRecurrentConnections)];
1725
1726
1727
1728 % Check for solution
1729 function flagSolution = solution_check(maxFitness, fitnessLimit, flagSolution)
1730
1731 % Check for solution
1732 if maxFitness > fitnessLimit
1733 flagSolution = 1;
1734 end
1735
1736 end
1737
1738 %% visualization()
1739 function visualization(figHandle, AxHandle, nAverageNonDisabledConnections, ...
1740 nAverageHiddenNodes, maxOverallFitness, nAverageDisabledConnections, ...
1741 nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie)
1742
1743 nGenerations = numel(nSpeciesArray);
1744
1745 lineSpec = {'LineWidth',1};
1746 fontSize = 8;
1747 nTick = 9;
1748 if 1 <= nGenerations && nGenerations < 6
1749 XTick1 = unique(round(linspace(1,5,5)));
1750 XTick2 = unique(round(linspace(1,5,5)));
1751 elseif 6 <= nGenerations && nGenerations < 10
1752 XTick1 = unique(round(linspace(1,nGenerations,9)));
1753 XTick2 = unique(round(linspace(1,nGenerations,6)));
1754 elseif 10 <= nGenerations && nGenerations < Inf
1755 XTick1 = unique(round(linspace(1,nGenerations,15)));
1756 XTick2 = unique(round(linspace(1,nGenerations,6)));
1757 end
1758 XTickLabel1 = cellfun(@num2str,num2cell(XTick1(:)), 'uniformoutput',false);
1759 XTickLabel2 = cellfun(@num2str,num2cell(XTick2(:)), 'uniformoutput',false);
1760
1761 % Maximum fitness
1762 setfigHandle, 'CurrentAxes', AxHandle.subPlot1);
1763 hold(AxHandle.subPlot1,'on');
1764 plot(AxHandle.subPlot1, maxOverallFitness(2,:), maxOverallFitness(1,:), lineSpec{:});
1765 xlabel(AxHandle.subPlot1,'Generation', 'FontSize', fontSize);
1766 ylabel(AxHandle.subPlot1,'Maximum fitness', 'FontSize', fontSize);
1767 grid(AxHandle.subPlot1,'on');
1768 hold(AxHandle.subPlot1,'off');
1769
1770 % Average fitness
1771 setfigHandle, 'CurrentAxes', AxHandle.subPlot2);
1772 hold(AxHandle.subPlot2,'on');
1773 plot(AxHandle.subPlot2, meanOverallFitness(2,:), meanOverallFitness(1,:), lineSpec{:});
1774 xlabel(AxHandle.subPlot2,'Generation', 'FontSize', fontSize);
1775 ylabel(AxHandle.subPlot2,'Average fitness', 'FontSize', fontSize);
1776 grid(AxHandle.subPlot2,'on');
1777 hold(AxHandle.subPlot2,'off');
1778
1779 % Average number of hidden nodes
1780 setfigHandle, 'CurrentAxes', AxHandle.subPlot3);
1781 hold(AxHandle.subPlot3,'on');
1782 plot(AxHandle.subPlot3, nAverageHiddenNodes(2,:), nAverageHiddenNodes(1,:), lineSpec{:});
1783 xlabel(AxHandle.subPlot3,'Generation', 'FontSize', fontSize);
1784 ylabel(AxHandle.subPlot3,'Avg. number of hidden nodes', 'FontSize', fontSize);
1785 grid(AxHandle.subPlot3,'on');
1786 hold(AxHandle.subPlot3,'off');
1787
1788 % Average number of non disabled connections
1789 setfigHandle, 'CurrentAxes', AxHandle.subPlot4);
1790 hold(AxHandle.subPlot4,'on');
1791 plot(AxHandle.subPlot4, nAverageNonDisabledConnections(2,:), ...
1792 nAverageNonDisabledConnections(1,:), lineSpec{:});
1793 xlabel(AxHandle.subPlot4,'Generation', 'FontSize', fontSize);
1794 ylabel(AxHandle.subPlot4,'Avg. number of non disabled connections', ...
1795 'FontSize', fontSize);
1796 grid(AxHandle.subPlot4,'on');
1797 hold(AxHandle.subPlot4,'off');
1798
1799 % Average number of disabled connections
1800 setfigHandle, 'CurrentAxes', AxHandle.subPlot5);
1801 hold(AxHandle.subPlot5,'on');
1802 plot(AxHandle.subPlot5, nAverageDisabledConnections(2,:), ...
1803 nAverageDisabledConnections(1,:), lineSpec{:});
1804 xlabel(AxHandle.subPlot5,'Generation', 'FontSize', fontSize);
1805 ylabel(AxHandle.subPlot5,'Avg. number of disabled connections', ...
1806 'FontSize', fontSize);
1807 grid(AxHandle.subPlot5,'on');
1808 hold(AxHandle.subPlot5,'off');
1809
1810 % Average number of recurrent connections
1811 setfigHandle, 'CurrentAxes', AxHandle.subPlot6);
1812 hold(AxHandle.subPlot6,'on');
1813 plot(AxHandle.subPlot6, nAverageDisabledConnections(2,:), ...
1814 nRecurrentConnection, lineSpec{:});
1815 xlabel(AxHandle.subPlot6,'Generation', 'FontSize', fontSize);
1816 ylabel(AxHandle.subPlot6,'Avg. number of recurrent connections', ...
1817 'FontSize', fontSize);
1818 grid(AxHandle.subPlot6,'on');
1819 hold(AxHandle.subPlot6,'off');
1820
1821 % Number of species
1822 setfigHandle, 'CurrentAxes', AxHandle.subPlot7);
1823 hold(AxHandle.subPlot7,'on');
1824 plot(AxHandle.subPlot7, nAverageDisabledConnections(2,:), nSpeciesArray, lineSpec{:});
1825 xlabel(AxHandle.subPlot7,'Generation', 'FontSize', fontSize);
1826 ylabel(AxHandle.subPlot7,'Number of species', ...
1827 'FontSize', fontSize);
1828 grid(AxHandle.subPlot7,'on');
1829 hold(AxHandle.subPlot7,'off');
1830
1831 % Average number of individuals per specie
1832 setfigHandle, 'CurrentAxes', AxHandle.subPlot8);
1833 hold(AxHandle.subPlot8,'on');
1834 plot(AxHandle.subPlot8, nAverageDisabledConnections(2,:), nIndPerSpecie, lineSpec{:});
1835 xlabel(AxHandle.subPlot8,'Generation', 'FontSize', fontSize);
1836 ylabel(AxHandle.subPlot8,'Avg. number of individuals per specie', ...
1837 'FontSize', fontSize);
1838 grid(AxHandle.subPlot8,'on');
1839 hold(AxHandle.subPlot8,'off');
1840
1841 drawnow;
1842 hold off
1843 end
1844
1845 end
1846
1847 %% print_generation()
1848 function print_generation(iGeneration)
1849
1850 fprintf([
1851 '%\n-----\n'
1852 ' Generation %d\n'
1853 '-----\n',
1854 i, iGeneration);
1855
1856 end
1857
1858 %% shrink_mutation()
1859 function Mutation = shrink_mutation(Mutation, iGeneration)
1860
1861 % Add node
1862 if Mutation.probabilityAddNodeScheduling(1) == -1
1863 for iGen = 1:numel(Mutation.probabilityAddNodeScheduling(1,:))
1864 if iGeneration < Mutation.probabilityAddNodeScheduling(1,iGen)
1865 Mutation.probabilityAddNode = Mutation.probabilityAddNodeScheduling(2,iGen);
1866 break;
1867 end
1868 end
1869 else
1870 Mutation.probabilityAddNode = max([
1871 Mutation.probabilityAddNode - Mutation.probabilityAddNodeShrinkRate, ...
1872 Mutation.probabilityAddNodeMin]);
1873 end
1874
1875 % Add connection
1876 if Mutation.probabilityAddConnectionScheduling(1) == -1
1877 for iGen = 1:numel(Mutation.probabilityAddConnectionScheduling(1,:))
1878 if iGeneration < Mutation.probabilityAddConnectionScheduling(1,iGen)
1879 Mutation.probabilityAddConnection = Mutation.probabilityAddConnectionScheduling(2,iGen);
1880 break;
1881 end
1882 else
1883 Mutation.probabilityAddConnection = max([
1884 Mutation.probabilityAddConnection - Mutation.probabilityAddConnectionShrinkRate, ...
1885 Mutation.probabilityAddConnectionMin]);
1886 end
1887
1888 % Recurrent connection
1889 Mutation.probabilityRecurrency = max([
1890 Mutation.probabilityRecurrency - Mutation.probabilityRecurrencyShrinkRate, ...
1891 Mutation.probabilityRecurrencyMin]);
1892
1893 % Weight mutation
1894 Mutation.probabilityMutateWeight = max([
1895 Mutation.probabilityMutateWeight - Mutation.probabilityMutateWeightShrinkRate, ...
1896 Mutation.probabilityMutateWeightMin]);
1897
1898 % Display new mutation probabilities
1899 fprintf([
1900 'Mutation probabilities update:\n',...
1901 'Mutation.probabilityAddNode = %.5f\n',...
1902 'Mutation.probabilityAddConnection = %.5f\n',...
1903 'Mutation.probabilityRecurrency = %.5f\n',...
1904 'Mutation.probabilityMutateWeight = %.5f\n',...
1905 'Mutation.probabilityMutateWeightMin = %.5f\n',...
1906 'Mutation.probabilityAddNode...',...
1907 'Mutation.probabilityAddConnection...',...
1908 'Mutation.probabilityRecurrency,...',...
1909 'Mutation.probabilityMutateWeight...',...
1910 'Mutation.probabilityMutateWeightMin = %.5f\n',...
1911 ]);
1912 end
1913 end

```

D. Evaluation Function

```

1 function [PopulationWithFitnesses, SimDataOut] = ...
2 usv_evaluation_function(Population, EvalFunParameters, fitnessLimit, SimData)
3
4 %% Init
5
6 % It seems that initializing the SimData structure inside
7 % usv_evaluation_function() causes problem in parfor simulation (i.e. fitness
8 % are all the same)
9 SimData = SimData;
10
11 % We store Population in PopulationWithFitnesses
12 PopulationWithFitnesses = Population;
13
14 % Number of individuals
15 nIndividuals = size(Population, 2);
16
17 % Number of inputs/outputs excluding previous layers' states
18 nInputs = 26;
19 nOutputs = 4;
20
21 % Create temporary directory to put the generated neural networks if it doesn't
22 % already exist
23 folderName = 'temp_usv_nn_controller';
24 if ~exist(folderName, 'dir')
25 mkdir(folderName);
26 end
27
28 % Number of trajectories
29 nTrajectories = numel(EvalFunParameters.Trajectory);
30
31 % Current generation number
32 iGeneration = EvalFunParameters.iGeneration;
33
34 %% Generate neural network functions for each individual
35
36 % Function name string
37 funNameString = 'usv_nn_controller';
38
39 % Folder name initialization
40 subFolderNameArray = cell(1,nIndividuals);
41 folderWithSubFolderName = cell(1,nIndividuals);
42 folderWithSubFolderAndFileWExtName = cell(1,nIndividuals);
43 folderWithSubFolderAndFileNExtName = cell(1,nIndividuals);
44 workerDir = cell(1,nIndividuals);
45
46 % Main directory
47 mainDir = strrep(pwd(), '\\', '/');
48
49 % Temporary subfolder. This helps avoid fuck up when using parfor
50 for individual = 1:nIndividuals
51 subFolderNameArray(individual) = gen_temp_string(30);
52 end
53
54 % We create an array to log which individual failed to have their network built.
55 % This is to make the code more robust to errors. A 1 means the individual is ok

```

```

56 % and a zero means it is not.
57 statusIndividualRecord = ones(1,nIndividuals);
58
59 % We create the temporary directories and generate the neural network functions
60 parfor iIndividual = 1:nIndividuals
61
62 % Shuffle random number generator seed
63 rng('shuffle');
64
65 % Worker directory
66 workerDir(iIndividual) = strrep(pwd,'\', '/');
67
68 % cd to main directory
69 cd(mainDir);
70
71 % Build neural network of current individual
72 try
73 [neuralNet, nDelays] = build_neural_network(Population(iIndividual));
74 catch
75 statusIndividualRecord(iIndividual) = 0;
76 fprintf(' Neural network build failed for individual % 4.f in generation % 4.f\n', ...
77 iIndividual, iGeneration);
78 end
79
80 % Folder/subFolder path
81 folderWithSubFolderName(iIndividual) = ...
82 {[folderName,'/',subFolderNameArray(iIndividual)]};
83
84 % Folder/subFolder/file without extension path
85 folderWithSubFolderAndFileWOExtName(iIndividual) = ...
86 {[folderWithSubFolderName(iIndividual), '/',funNameString];
87
88 % Folder/subFolder/file with extension path
89 folderWithSubFolderAndFileWExtName(iIndividual) = ...
90 {[folderWithSubFolderAndFileWOExtName(iIndividual), '.m']};
91
92 % Add temporary folder to path
93 cd(mainDir,'/');
94 mkdir(subFolderNameArray(iIndividual));
95 cd(mainDir);
96 addpath(folderWithSubFolderName(iIndividual));
97
98 % Generate function
99 if statusIndividualRecord(iIndividual)
100    genFunctionMod(neuralNet, ...
101        folderWithSubFolderAndfileWOExtName(iIndividual), ...
102        'MatrixOnly', 'yes', 'ShowLinks', 'no');
103 end
104
105 % Rehash (https://goo.gl/IDtpzk)
106 exist([funNameString,'.m'],'file');
107 rehash();
108
109 % Remove path
110 rmpath(folderWithSubFolderName(iIndividual));
111 rehash();
112
113 % cd back to worker directory
114 cd(workerDir(iIndividual));
115
116 end
117
118 % We add a little pause in order for the filesystem to record to new created files
119 pause(1);
120
121 % We then modify the generated function. Initially, this was in the same parfor
122 % loop above but it was not reliable (don't know why). Moving this step in a
123 % separate parfor loop seems to have helped avoid problem.
124 nPreviousStatesArray = zeros(1,nIndividuals);
125 parfor iIndividual = 1:nIndividuals
126
127 if statusIndividualRecord(iIndividual)
128    % Shuffle random number generator seed
129    rng('shuffle');
130
131 % Worker directory
132 workerDir(iIndividual) = strrep(pwd,'\', '/');
133
134 % cd to main directory
135 cd(mainDir);
136
137 % Add temporary folder to path
138 addpath(folderWithSubFolderName(iIndividual));
139
140 % Create function handle
141 funHandle = str2func(funNameString);
142
143 % https://goo.gl/IDtpzk
144 exist([funNameString,'.m'],'file');
145 rehash();
146
147 % Number of previous states
148 nPreviousStates = nargin(funHandle) - nInputs;
149 if nPreviousStates > 0
150    fprintf('nPreviousStates = % 4.f for individual % 4.f in generation % 4.f (%s)\n', ...
151        nPreviousStates, iIndividual, iGeneration, subFolderNameArray(iIndividual));
152 end
153 nPreviousStatesArray(iIndividual) = nPreviousStates;
154
155 % Post-process generated function
156 generated_mn_processing(...);
157 folderWithSubFolderAndfileWExtName(iIndividual), ...
158 funNameString, nPreviousStates);
159
160 % Remove path
161 rmpath(folderWithSubFolderName(iIndividual));
162 rehash();
163
164 % cd back to worker directory
165 cd(workerDir(iIndividual));
166
167 end
168
169 pause(1);
170
171 %% Evaluate networks
172
173 % To avoid broadcasting the entire structure into the parfor loop
174 Trajectory = EvalFunParameters.Trajectory;
175 SimParam = EvalFunParameters.SimParam;
176 UavParam = EvalFunParameters.UavParam;
177
178 xMin = UavParam.xMin;
179 xMax = UavParam.xMax;
180 yMin = UavParam.yMin;
181 yMax = UavParam.yMax;
182 zMin = UavParam.zMin;
183 zMax = UavParam.zMax;
184 psiMin = UavParam.psiMin;
185 psiMax = UavParam.psiMax;
186 thetaMin = UavParam.thetaMin;
187 thetaMax = UavParam.thetaMax;
188 phiMin = UavParam.phiMin;
189 phiMax = UavParam.phiMax;
190
191 % Compute maximum time fitness
192 timeCellArray = (Trajectory(:,XDesired));
193 timeVector = zeros(1,nTrajectories);
194 for iTraj = 1:nTrajectories
195    timeVector(iTraj) = timeCellArray(iTraj).t(end);
196 end
197 maxTimeFitness = sum(timeVector);
198
199 % Model name
200 modelName = 'uav_closed_loop_v5';
201
202 % https://goo.gl/uNCs79 (In parfor, how to create a temp folder for every worker)
203 if EvalFunParameters.useParallel
204
205 %% Parallel evaluation
206
207 % Load model
208 load_system(modelName);
209
210 % Cycle through population
211 parfor iIndividual = 1:nIndividuals
212
213 % We only simulate if the individual is ok
214 if statusIndividualRecord(iIndividual)
215
216 % Shuffle random number generator seed
217 rng('shuffle');
218
219 % Current work directory
220 cwd = strrep(pwd,'\', '/');
221
222 % Add current work directory in current session
223 addpath(cwd);
224
225 % Create temporary folder and change current directory to this folder
226 tmpdir = strrep(tempname,'\', '/');
227 mkdir(tmpdir);
228 cd(tmpdir);
229
230 % Add path to uav_nn_controller
231 addpath([mainDir,'\', folderWithSubFolderName(iIndividual)]);
232 rehash();
233
234 % Load system
235 load_system(modelName);
236 % load_system(modelName);
237
238 % Create function handle
239 funHandle = str2func(funNameString);
240
241 % Initial state of recurrent layer
242 if nPreviousStatesArray(iIndividual) == 0
243    initialState = 0;
244 else
245    initialState = zeros(1,nPreviousStatesArray(iIndividual));
246 end
247
248 % Neural network Id
249 neuralNetId = iIndividual;
250
251 % Initialize cell array that will contain simulation results
252 t = cell(1,nTrajectories);
253 xDesired = cell(1,nTrajectories);
254 yDesired = cell(1,nTrajectories);
255 zDesired = cell(1,nTrajectories);
256 psiDesired = cell(1,nTrajectories);
257 thetaDesired = cell(1,nTrajectories);
258 phiDesired = cell(1,nTrajectories);
259 x = cell(1,nTrajectories);
260 y = cell(1,nTrajectories);
261 z = cell(1,nTrajectories);
262 psi = cell(1,nTrajectories);
263 theta = cell(1,nTrajectories);
264 phi = cell(1,nTrajectories);
265 Va1 = cell(1,nTrajectories);
266 Va2 = cell(1,nTrajectories);
267 Va3 = cell(1,nTrajectories);
268 Va4 = cell(1,nTrajectories);
269
270 % Simulate trajectories
271 rapidAccEn = false;
272 if strcmp(SimParam.solverType,'Variable-step')
273    set_param(modelName,...
274        'SolverType',SimParam.solverType',...
275        'Solver',SimParam.solver',...
276        'SimulationMode',SimParam.simulationMode,...
277        'InitialStep',SimParam.initialStep,...
278        'SaveFormat','StructureWithTime',...
279        'RelTol',SimParam.reltol...
280        'AbsTol',SimParam.absTol...
281        'MinStep',SimParam.minstep...
282        'MaxStep',SimParam.maxstep...
283        'MaxConsecutiveMinStep',SimParam.maxConsecutiveMinStep...
284        'SFSSimEcho','off',...
285        'SimIntegrity','off');
286    else
287        set_param(modelName,...
288            'SolverType',SimParam.solverType,...
289            'Solver',SimParam.solver',...
290            'SimulationMode',SimParam.simulationMode,...
291            'FixedStep',SimParam.fixedStep...
292            'SaveFormat','StructureWithTime',...
293            'RelTol',SimParam.reltol...
294            'AbsTol',SimParam.absTol...
295            'SFSSimEcho','off',...
296            'SimIntegrity','off');
297    end
298 if nPreviousStatesArray(iIndividual) > 0 && rapidAccEn
299    set_param(modelName, ...
300        'SimulationMode','rapid-accelerator');
301 end
302
303 % set_param(modelName,'SimCtrlIC','on');
304 for iTraj = 1:nTrajectories
305
306 % Assign simulation variables to base workspace
307 assignin('base','g', UavParam.g); %#ok
308 assignin('base','mMotors', UavParam.nMotors); %#ok
309 assignin('base','l', UavParam.l); %#ok
310 assignin('base','PP', UavParam.PP); %#ok
311 assignin('base','Ra', UavParam.Ra); %#ok
312 assignin('base','La', UavParam.La); %#ok
313 assignin('base','ke', UavParam.ke); %#ok
314 assignin('base','km', UavParam.km); %#ok
315 assignin('base','Jz', UavParam.Jz); %#ok
316 assignin('base','Jx', UavParam.Jx); %#ok
317 assignin('base','Bz', UavParam.Bz); %#ok
318 assignin('base','kT', UavParam.kT); %#ok
319 assignin('base','VaMin', UavParam.VaMin); %#ok
320 assignin('base','VaMax', UavParam.VaMax); %#ok
321 assignin('base','omegalInitial', Trajectory(iTraj).IniCond.omegalInitial); %#ok
322 assignin('base','omegdInitial', Trajectory(iTraj).IniCond.omegdInitial); %#ok
323 assignin('base','iInitial', Trajectory(iTraj).IniCond.iInitial); %#ok
324 assignin('base','iaInitial', Trajectory(iTraj).IniCond.iaInitial); %#ok
325 assignin('base','iadInitial', Trajectory(iTraj).IniCond.iadInitial); %#ok
326 assignin('base','vptInitial', Trajectory(iTraj).IniCond.vptInitial); %#ok
327 assignin('base','xInitial', Trajectory(iTraj).IniCond.xInitial); %#ok
328 assignin('base','xDelay', Trajectory(iTraj).IniCond.xDelay); %#ok
329 assignin('base','omegabInitial', Trajectory(iTraj).IniCond.omegabInitial); %#ok
330 assignin('base','omegabDelay', Trajectory(iTraj).IniCond.omegabDelay); %#ok
331 assignin('base','euler2XYZInitial', Trajectory(iTraj).IniCond.euler2XYZInitial); %#ok
332 assignin('base','euler2XYZDelay', Trajectory(iTraj).IniCond.euler2XYZDelay); %#ok
333 assignin('base','vDesired', Trajectory(iTraj).vDesired); %#ok
334 assignin('base','lDesired', Trajectory(iTraj).lDesired); %#ok
335 assignin('base','psiDesired', Trajectory(iTraj).psiDesired); %#ok
336 assignin('base','ThetaDesired', Trajectory(iTraj).ThetaDesired); %#ok
337 assignin('base','PhiDesired', Trajectory(iTraj).PhiDesired); %#ok
338 assignin('base','neuralNetworkId', neuralNetworkId); %#ok
339

```

```

340 assignIn('base', 'nPreviousStates', nPreviousStatesArray(iIndividual)); %ok
341 assignIn('base', 'initialState', initialState); %ok
342 assignIn('base', 'funNameString', double(funNameString)); %ok
343 assignIn('base', 'lowerXTol', Trajectory(iTraj).XDesired.tol(1)); %ok
344 assignIn('base', 'upperXTol', Trajectory(iTraj).XDesired.tol(2)); %ok
345 assignIn('base', 'lowerYtol', Trajectory(iTraj).YDesired.tol(1)); %ok
346 assignIn('base', 'upperYtol', Trajectory(iTraj).YDesired.tol(2)); %ok
347 assignIn('base', 'lowerZtol', Trajectory(iTraj).ZDesired.tol(1)); %ok
348 assignIn('base', 'upperZtol', Trajectory(iTraj).ZDesired.tol(2)); %ok
349 assignIn('base', 'lowerPsitol', Trajectory(iTraj).PsiDesired.tol(1)); %ok
350 assignIn('base', 'upperPsitol', Trajectory(iTraj).PsiDesired.tol(2)); %ok
351 assignIn('base', 'lowerThetitol', Trajectory(iTraj).ThetaDesired.tol(1)); %ok
352 assignIn('base', 'upperThetitol', Trajectory(iTraj).ThetaDesired.tol(2)); %ok
353 assignIn('base', 'lowerPhitol', Trajectory(iTraj).PhiDesired.tol(1)); %ok
354 assignIn('base', 'upperPhitol', Trajectory(iTraj).PhiDesired.tol(2)); %ok
355 assignIn('base', 'maxViolationX', Trajectory(iTraj).maxViolationX); %ok
356 assignIn('base', 'maxViolationY', Trajectory(iTraj).maxViolationY); %ok
357 assignIn('base', 'maxViolationZ', Trajectory(iTraj).maxViolationZ); %ok
358 assignIn('base', 'maxViolationPsi', Trajectory(iTraj).maxViolationPsi); %ok
359 assignIn('base', 'maxViolationTheta', Trajectory(iTraj).maxViolationTheta); %ok
360 assignIn('base', 'maxViolationPhi', Trajectory(iTraj).maxViolationPhi); %ok
361 assignIn('base', 'violationFactor', Trajectory(iTraj).violationFactor); %ok
362 assignIn('base', 'violationFactorX', Trajectory(iTraj).violationFactorX); %ok
363 assignIn('base', 'violationFactorY', Trajectory(iTraj).violationFactorY); %ok
364 assignIn('base', 'violationFactorZ', Trajectory(iTraj).violationFactorZ); %ok
365 assignIn('base', 'violationFactorPsi', Trajectory(iTraj).violationFactorPsi); %ok
366 assignIn('base', 'violationFactorTheta', Trajectory(iTraj).violationFactorTheta); %ok
367 assignIn('base', 'currentTimeThreshold', Trajectory(iTraj).violationTimeThreshold); %ok
368 assignIn('base', 'currentTimeThreshold1', Trajectory(iTraj).violationTimeThreshold1); %ok
369 assignIn('base', 'currentTimeThreshold1D', Trajectory(iTraj).violationTimeThreshold1D); %ok
370 assignIn('base', 'currentTimeThreshold1P', Trajectory(iTraj).violationTimeThreshold1P); %ok
371 assignIn('base', 'currentTimeThreshold1T', Trajectory(iTraj).violationTimeThreshold1T); %ok
372 assignIn('base', 'currentTimeThreshold2', Trajectory(iTraj).violationTimeThreshold2); %ok
373 assignIn('base', 'currentTimeThreshold2D', Trajectory(iTraj).violationTimeThreshold2D); %ok
374 assignIn('base', 'currentTimeThreshold2P', Trajectory(iTraj).violationTimeThreshold2P); %ok
375 assignIn('base', 'currentTimeThreshold2T', Trajectory(iTraj).violationTimeThreshold2T); %ok
376 assignIn('base', 'xDesiredMin', UavParam.xDesiredMin); %ok
377 assignIn('base', 'xDesiredMax', UavParam.xDesiredMax); %ok
378 assignIn('base', 'yDesiredMin', UavParam.yDesiredMin); %ok
379 assignIn('base', 'yDesiredMax', UavParam.yDesiredMax); %ok
380 assignIn('base', 'zDesiredMin', UavParam.zDesiredMin); %ok
381 assignIn('base', 'zDesiredMax', UavParam.zDesiredMax); %ok
382 assignIn('base', 'psiDesiredMin', UavParam.psiDesiredMin); %ok
383 assignIn('base', 'psiDesiredMax', UavParam.psiDesiredMax); %ok
384 assignIn('base', 'thetaDesiredMin', UavParam.thetaDesiredMin); %ok
385 assignIn('base', 'thetaDesiredMax', UavParam.thetaDesiredMax); %ok
386 assignIn('base', 'phiDesiredMin', UavParam.phiDesiredMin); %ok
387 assignIn('base', 'phiDesiredMax', UavParam.phiDesiredMax); %ok
388 assignIn('base', 'xMin', UavParam.xMin); %ok
389 assignIn('base', 'xMax', UavParam.xMax); %ok
390 assignIn('base', 'yMin', UavParam.yMin); %ok
391 assignIn('base', 'yMax', UavParam.yMax); %ok
392 assignIn('base', 'zMin', UavParam.zMin); %ok
393 assignIn('base', 'zMax', UavParam.zMax); %ok
394 assignIn('base', 'uMin', UavParam.uMin); %ok
395 assignIn('base', 'uMax', UavParam.uMax); %ok
396 assignIn('base', 'vMin', UavParam.vMin); %ok
397 assignIn('base', 'vMax', UavParam.vMax); %ok
398 assignIn('base', 'wMin', UavParam.wMin); %ok
399 assignIn('base', 'wMax', UavParam.wMax); %ok
400 assignIn('base', 'udotMin', UavParam.udotMin); %ok
401 assignIn('base', 'udotMax', UavParam.udotMax); %ok
402 assignIn('base', 'vdotMin', UavParam.vdotMin); %ok
403 assignIn('base', 'vdotMax', UavParam.vdotMax); %ok
404 assignIn('base', 'wdotMin', UavParam.wdotMin); %ok
405 assignIn('base', 'wdotMax', UavParam.wdotMax); %ok
406 assignIn('base', 'psiMin', UavParam.psiMin); %ok
407 assignIn('base', 'psiMax', UavParam.psiMax); %ok
408 assignIn('base', 'thetaMin', UavParam.thetaMin); %ok
409 assignIn('base', 'thetaMax', UavParam.thetaMax); %ok
410 assignIn('base', 'phiMin', UavParam.phiMin); %ok
411 assignIn('base', 'phiMax', UavParam.phiMax); %ok
412 assignIn('base', 'pMin', UavParam.pMin); %ok
413 assignIn('base', 'pMax', UavParam.pMax); %ok
414 assignIn('base', 'qMin', UavParam.qMin); %ok
415 assignIn('base', 'qMax', UavParam.qMax); %ok
416 assignIn('base', 'rMin', UavParam.rMin); %ok
417 assignIn('base', 'rMax', UavParam.rMax); %ok
418 assignIn('base', 'omegamin', UavParam.omegamin); %ok
419 assignIn('base', 'omegap1', UavParam.omegap1); %ok
420 assignIn('base', 'omegap2', UavParam.omegap2); %ok
421 assignIn('base', 'omegap3', UavParam.omegap3); %ok
422 assignIn('base', 'omegap4', UavParam.omegap4); %ok
423 assignIn('base', 'omegap4min', UavParam.omegap4min); %ok
424 assignIn('base', 'omegap4max', UavParam.omegap4max); %ok
425
426 % Launch simulation
427 try
428   if nPreviousStatesArray(iIndividual) == 0 || -rapidAccn
429     simout = sim(modelName,...)
430     StopTime = num2str(max(Trajectory(iTraj).XDesired.t));
431     %FastRestart = SimParam.fastRestart;
432   else
433     simout = sim(modelName,...)
434     'StopTime',num2str(max(Trajectory(iTraj).XDesired.t));
435   end
436 catch ME
437   statusIndividualRecord(iIndividual) = 0;
438   fprintf('Simulation failed for individual %s.f in generation %s.f. Error is:\n', ...
439           iIndividual, iGeneration);
440   fprintf('Identifier: %s message: %s\n', ME.identifier, ME.message);
441   break;
442 end
443
444 %% Extract simulation results
445 simout = simout.get('simout');
446 t{iTraj} = simout.time;
447 xDesired{iTraj} = simout.signals.values(:,1);
448 zDesired{iTraj} = simout.signals.values(:,2);
449 yDesired{iTraj} = simout.signals.values(:,3);
450 psiDesired{iTraj} = simout.signals.values(:,4);
451 thetaDesired{iTraj} = simout.signals.values(:,5);
452 phiDesired{iTraj} = simout.signals.values(:,6);
453 x{iTraj} = simout.signals.values(:,7);
454 y{iTraj} = simout.signals.values(:,8);
455 z{iTraj} = simout.signals.values(:,9);
456 psi{iTraj} = simout.signals.values(:,10);
457 theta{iTraj} = simout.signals.values(:,17);
458 phi{iTraj} = simout.signals.values(:,18);
459 Vai{iTraj} = simout.signals.values(:,41);
460 Vaz{iTraj} = simout.signals.values(:,42);
461 Va3{iTraj} = simout.signals.values(:,43);
462 Va4{iTraj} = simout.signals.values(:,44);
463
464 end
465 if nPreviousStatesArray(iIndividual) == 0 || -rapidAccn
466   set_param(modelName,'FastRestart','off');
467 end
468
469 % Close system
470 close_system(modelName_0);
471
472 % Remove path and folder to use_m_controller
473 rmpath([mainDir,'/',folderWithSubFolderName(iIndividual)]);
474 try
475   rmdir([mainDir,'/',folderWithSubFolderName(iIndividual)], 's');
476 catch
477   try
478     rmdir([mainDir,'/',folderWithSubFolderName(iIndividual)]);
479   catch
480     fprintf('Unable to remove %s for individual %s.f in generation %s.f\n', ...
481             [mainDir,'/',folderWithSubFolderName(iIndividual)], iIndividual, iGeneration);
482   end
483 end
484
485 %% Change back to work directory and remove temporary directory
486 cd(cwd);
487 try
488   rmdir(tmpdir, 's');
489 catch
490   try
491     rmdir(tmpdir);
492   catch
493     fprintf('Unable to remove %s for individual %s.f in generation %s.f\n', ...
494             tmpdir, iIndividual, iGeneration);
495   end
496 end
497 rmpath(cwd);
498
499 end
500
501 % Compute fitness for this individual
502 if statusIndividualRecord(iIndividual)
503   [fitnessSignal, rewardTime] = compute_signal_error_fitness(nTrajectories, maxTimeFitness, ...
504     xMin, xMax, yMin, yMax, zMin, zMax, psiMin, psiMax, thetaMin, thetaMax, phiMin, phiMax, ...
505     t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
506     x, y, z, psi, theta, phi, Val, Va2, Va3, Va4);
507   fitness = compute_fitness(nTrajectories, Trajectory, ...
508     t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
509     x, y, z, psi, theta, phi, rewardTime);
510   PopulationWithFitnesses(iIndividual).fitness = fitness * fitnessSignal;
511 else
512   PopulationWithFitnesses(iIndividual).fitness = 0;
513 end
514
515 fprintf('fitness = %s.2f for individual %s.f in generation %s.f\n', ...
516         PopulationWithFitnesses(iIndividual).fitness, iIndividual, iGeneration);
517
518 % Store simulation results
519 if statusIndividualRecord(iIndividual)
520   for iTraj = 1:nTrajectories
521     nPtsTemp = numel(t{iTraj});
522     SimDataOut(iIndividual).x{iTraj}(1:nPtsTemp) = x{iTraj};
523     SimDataOut(iIndividual).y{iTraj}(1:nPtsTemp) = y{iTraj};
524     SimDataOut(iIndividual).z{iTraj}(1:nPtsTemp) = z{iTraj};
525     SimDataOut(iIndividual).psi{iTraj}(1:nPtsTemp) = psi{iTraj};
526     SimDataOut(iIndividual).theta{iTraj}(1:nPtsTemp) = theta{iTraj};
527     SimDataOut(iIndividual).phi{iTraj}(1:nPtsTemp) = phi{iTraj};
528     SimDataOut(iIndividual).thetaDesired{iTraj}(1:nPtsTemp) = thetaDesired{iTraj};
529     SimDataOut(iIndividual).phiDesired{iTraj}(1:nPtsTemp) = phiDesired{iTraj};
530     SimDataOut(iIndividual).x{iTraj}(1:nPtsTemp) = x{iTraj};
531     SimDataOut(iIndividual).y{iTraj}(1:nPtsTemp) = y{iTraj};
532     SimDataOut(iIndividual).z{iTraj}(1:nPtsTemp) = z{iTraj};
533     SimDataOut(iIndividual).psi{iTraj}(1:nPtsTemp) = psi{iTraj};
534     SimDataOut(iIndividual).theta{iTraj}(1:nPtsTemp) = theta{iTraj};
535     SimDataOut(iIndividual).phi{iTraj}(1:nPtsTemp) = phi{iTraj};
536     SimDataOut(iIndividual).Val{iTraj}(1:nPtsTemp) = Val{iTraj};
537     SimDataOut(iIndividual).Va2{iTraj}(1:nPtsTemp) = Va2{iTraj};
538     SimDataOut(iIndividual).Va3{iTraj}(1:nPtsTemp) = Va3{iTraj};
539     SimDataOut(iIndividual).Va4{iTraj}(1:nPtsTemp) = Va4{iTraj};
540     SimDataOut(iIndividual).t{iTraj} = t{iTraj};
541     SimDataOut(iIndividual).x{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).t{iTraj}(1:nPtsTemp);
542     SimDataOut(iIndividual).y{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).x{iTraj}(1:nPtsTemp);
543     SimDataOut(iIndividual).z{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).y{iTraj}(1:nPtsTemp);
544     SimDataOut(iIndividual).psi{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).z{iTraj}(1:nPtsTemp);
545     SimDataOut(iIndividual).theta{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).psi{iTraj}(1:nPtsTemp);
546     SimDataOut(iIndividual).phi{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).theta{iTraj}(1:nPtsTemp);
547     SimDataOut(iIndividual).thetaDesired{iTraj} = SimDataOut(iIndividual).phiDesired{iTraj}(1:nPtsTemp);
548     SimDataOut(iIndividual).phiDesired{iTraj} = SimDataOut(iIndividual).phiDesired{iTraj}(1:nPtsTemp);
549     SimDataOut(iIndividual).x{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).phiDesired{iTraj}(1:nPtsTemp);
550     SimDataOut(iIndividual).y{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).x{iTraj}(1:nPtsTemp);
551     SimDataOut(iIndividual).z{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).y{iTraj}(1:nPtsTemp);
552     SimDataOut(iIndividual).psi{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).z{iTraj}(1:nPtsTemp);
553     SimDataOut(iIndividual).theta{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).psi{iTraj}(1:nPtsTemp);
554     SimDataOut(iIndividual).phi{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).theta{iTraj}(1:nPtsTemp);
555     SimDataOut(iIndividual).Val{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).Val{iTraj}(1:nPtsTemp);
556     SimDataOut(iIndividual).Va2{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).Val2{iTraj}(1:nPtsTemp);
557     SimDataOut(iIndividual).Va3{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).Val3{iTraj}(1:nPtsTemp);
558     SimDataOut(iIndividual).Va4{iTraj}(1:nPtsTemp) = SimDataOut(iIndividual).Val4{iTraj}(1:nPtsTemp);
559   end
560 end
561
562 else % Don't use parallel
563
564 %% Serial evaluation
565
566 % Load system (just once)
567 load_system(modelName);
568
569 % Cycle through population
570 for iIndividual = 1:nIndividuals
571
572   % We only simulate if the individual is ok
573   if statusIndividualRecord(iIndividual)
574
575     % Add temporary folder in path
576     addpath(folderWithSubFolderName(iIndividual));
577     rehash();
578
579     % Create function handle
580     funHandle = str2func(funNameString);
581
582     % Initial state of recurrent layer
583     if nPreviousStatesArray(iIndividual) == 0
584       initialState = zeros(1,nPreviousStatesArray(iIndividual));
585     else
586       initialState = initialState(iIndividual);
587     end
588
589     % Neural network ID
590     neuralNetworkID = iIndividual;
591
592     % Initialize cell array that will contain simulation results
593     t = cell(1,nTrajectories);
594     xDesired = cell(1,nTrajectories);
595     yDesired = cell(1,nTrajectories);
596     zDesired = cell(1,nTrajectories);
597     psiDesired = cell(1,nTrajectories);
598     thetaDesired = cell(1,nTrajectories);
599     phiDesired = cell(1,nTrajectories);
600     x = cell(1,nTrajectories);
601     y = cell(1,nTrajectories);
602     z = cell(1,nTrajectories);
603     psi = cell(1,nTrajectories);
604     theta = cell(1,nTrajectories);
605     phi = cell(1,nTrajectories);
606     Val = cell(1,nTrajectories);
607     Val2 = cell(1,nTrajectories);
608     Val3 = cell(1,nTrajectories);
609     Val4 = cell(1,nTrajectories);
610
611     % Simulate trajectories
612     rapidAccn = true;
613     set_param(modelName,'FastRestart','off');
614     if strcmp(SimParam.solverType,'Variable-step')
615       set_param(modelName,'VariableStep');
616       SolverType = SimParam.solverType;
617       Solver = SimParam.solver;
618       SimulationMode = SimParam.simulationMode;
619       InitialStep = SimParam.initialStep;
620       SaveFormat = SimParam.structureWithTime;
621       RelTol = SimParam.reltol;
622       AbsTol = SimParam.absTol;
623
624   end

```

```

624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907

```

```

908 Trajectory(iTraj).XDesired.tol(2), t(iTraj), xDesired(iTraj));
909 [yLower, yUpper] = compute_lower_upper_bound(Trajectory(iTraj).YDesired.tol(1), ...
910 Trajectory(iTraj).YDesired.tol(2), t(iTraj), yDesired(iTraj));
911 [zLower, zUpper] = compute_lower_upper_bound(Trajectory(iTraj).ZDesired.tol(1), ...
912 Trajectory(iTraj).ZDesired.tol(2), t(iTraj), zDesired(iTraj));
913
914 [psiLower, psiUpper] = compute_lower_upper_bound(Trajectory(iTraj).PsiDesired.tol(1), ...
915 Trajectory(iTraj).PsiDesired.tol(2), t(iTraj), psiDesired(iTraj));
916
917 [thetaLower, thetaUpper] = compute_lower_upper_bound(Trajectory(iTraj).ThetaDesired.tol(1), ...
918 Trajectory(iTraj).ThetaDesired.tol(2), t(iTraj), thetaDesired(iTraj));
919
920 [phiLower, phiUpper] = compute_lower_upper_bound(Trajectory(iTraj).PhiDesired.tol(1), ...
921 Trajectory(iTraj).PhiDesired.tol(2), t(iTraj), phiDesired(iTraj));
922
923
924 % Compute the total duration that each trajectory is within the tolerance
925 durationX = sum(deltaT((zLower <= x(iTraj)) & (x(iTraj) <= xUpper)));
926 durationY = sum(deltaT((yLower <= y(iTraj)) & (y(iTraj) <= yUpper)));
927 durationZ = sum(deltaT((zLower <= z(iTraj)) & (z(iTraj) <= zUpper)));
928 durationPsi = sum(deltaT((psiLower <= psi(iTraj)) & (psi(iTraj) <= psiUpper)));
929 durationTheta = sum(deltaT((thetaLower <= theta(iTraj)) & (theta(iTraj) <= thetaUpper)));
930 durationPhi = sum(deltaT((phiLower <= phi(iTraj)) & (phi(iTraj) <= phiUpper)));
931
932
933 % Compute total fitness
934 f = f + rewardTime(iTraj)*(durationX + durationY + durationZ + durationPsi + durationTheta + durationPhi);
935
936 end
937
938 end
939
940
941 %% Function to compute the fitness associated to the trajectory error
942 function [fitnessSignal, rewardTime] = compute_signal_error_fitness(nTrajectories, maxTimeFitness, ...
943 xMin, xMax, yMin, yMax, zMin, zMax, psiMin, psiMax, thetaMin, thetaMax, phiMin, phiMax, ...
944 t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
945 x, y, z, psi, theta, phi, Va1, Va2, Va3, Va4)
946
947
948 % Initialize total fitness to zero
949 fitness = 0;
950 fitnessSignal = 0;
951 fitnessSignalMax = 0;
952 rewardTime = zeros(1,nTrajectories);
953
954 % Compute fitness for each trajectory
955 for iTraj = 1:nTrajectories
956
957 % Compute fitnesses
958 [fitnessSignalX, fitnessMaxX] = sub_fun_error_fitness(xMin, xMax, xDesired(iTraj), x(iTraj));
959 [fitnessSignalY, fitnessMaxY] = sub_fun_error_fitness(yMin, yMax, yDesired(iTraj), y(iTraj));
960 [fitnessSignalZ, fitnessMaxZ] = sub_fun_error_fitness(zMin, zMax, zDesired(iTraj), z(iTraj));
961 [fitnessSignalPsi, fitnessMaxPsi] = sub_fun_error_fitness(psiMin, psiMax, psiDesired(iTraj), psi(iTraj));
962 [fitnessSignalTheta, fitnessMaxTheta] = sub_fun_error_fitness(thetaMin, thetaMax, thetaDesired(iTraj), theta(iTraj));
963 [fitnessSignalPhi, fitnessMaxPhi] = sub_fun_error_fitness(phiMin, phiMax, phiDesired(iTraj), phi(iTraj));
964
965 % Conditions to penalize controllers that send very low or very high voltages
966 VaMinThreshold = 1;
967 VaMaxThreshold = 100;
968 VaMinRatioThreshold = 0.80;
969 VaMaxRatioThreshold = 2.0;
970 VaMaxTiesThreshold = 0.8;
971 VaMaxRatioTiesThreshold = 0.80;
972 VaMinTiesThreshold = 2;
973 [condVaMin, condVaMax] = check_cond_Va(t(iTraj), Va1(iTraj), Va2(iTraj), Va3(iTraj), Va4(iTraj), ...
974 VaMinThreshold, VaMinRatioThreshold, VaMinTimeThreshold, ...
975 VaMaxThreshold, VaMaxRatioThreshold, VaMaxTimeThreshold);
976
977 % Conditions to penalize controllers that let the altitude decrease or increase
978 % sharply in Z and/or psi. Compute gradient of signalErrorZ and signalErrorPsi;
979 minErrRatioThrsld = -0.80;
980 ersZThrsld = -0.5;
981 maxErrZThrsld = 2;
982 ersPaiThrsld = -4*pi/180;
983 maxErrPaiThrsld = 8*pi/180;
984 gradErrZ = -gradient(signalErrorSignedZ);
985 gradErrPsi = -gradient(signalErrorSignedPsi);
986 [condGradPosZ, condGradNegZ] = check_cond_grad(gradErrZ, signalErrorSignedZ, ...
987 errZThrsld, maxErrZThrsld, minErrRatioThrsld);
988 [condGradPosPsi, condGradNegPsi] = check_cond_grad(gradErrPsi, signalErrorSignedPsi, ...
989 ersPaiThrsld, maxErrPaiThrsld, minErrRatioThrsld);
990
991
992 % Compute rewards
993 overallCondGrad = condGradPosZ || condGradNegZ || condGradPosPsi || condGradNegPsi;
994 [rewardTime(iTraj), rewardZ, rewardPsi] = compute_reward(overallCondGrad, ...
995 condGradPosZ, condGradNegZ, condGradPosPsi, condGradNegPsi, t(iTraj));
996
997 % Compute total fitness
998 if overallCondGrad && -condVaMin && -condVaMax
999 fitnessSignal = fitnessSignalX + fitnessSignalY + fitnessSignalZ + ...
1000 rewardZ*fitnessSignalZ + rewardPsi*fitnessSignalPsi + ...
1001 fitnessSignalTheta + fitnessSignalPhi;
1002 fitnessSignalMax = fitnessSignalMax + fitnessMaxX + fitnessMaxY + ...
1003 fitnessMaxZ + fitnessMaxPsi + fitnessMaxTheta + fitnessMaxPhi;
1004 elseif condVaMin || condVaMax
1005 fitnessSignal = fitnessSignalMax + fitnessMaxX + fitnessMaxY + ...
1006 fitnessMaxZ + fitnessMaxPsi + fitnessMaxTheta + fitnessMaxPhi;
1007 else
1008 fitnessSignal = fitnessSignal + fitnessSignalX + fitnessSignalY + ...
1009 rewardZ*fitnessSignalZ + rewardPsi*fitnessSignalPsi + ...
1010 fitnessSignalTheta + fitnessSignalPhi;
1011 fitnessSignalMax = fitnessSignalMax + fitnessMaxX + fitnessMaxY + ...
1012 fitnessMaxZ + fitnessMaxPsi + fitnessMaxTheta + fitnessMaxPhi;
1013 end
1014
1015
1016 % Fitness scaling
1017 fitnessSignal = maxTimeFitness*fitnessSignal/fitnessSignalMax;
1018
1019 function [fitnessSignal, fitnessMax, signalErrorSigned] = sub_fun_error_fitness(qMin, qMax, qDesired, q)
1020 % Compute max possible error
1021 sumSignalError1 = sum(qDesired - qMin.*ones(size(qDesired)));
1022 sumSignalError2 = sum(qDesired - qMax.*ones(size(qDesired)));
1023 sumSignalErrorMax = max([sumSignalError1,sumSignalError2]);
1024 % Compute actual error
1025 signalErrorSigned = q - qDesired;
1026 signalError = abs(signalErrorSigned);
1027 sumSignalError = sum(signalError);
1028 % Fitness
1029 fitnessSignal = abs(sumSignalErrorMax - sumSignalError);
1030 fitnessMax = abs(sumSignalErrorMax);
1031
1032
1033 function [condVaMin, condVaMax] = check_cond_Va(t,Va1,Va2,Va3,Va4, ...
1034 VaMinThreshold, VaMinRatioThreshold, VaMinTimeThreshold, ...
1035 VaMaxThreshold, VaMaxRatioThreshold, VaMaxTimeThreshold)
1036
1037 % Conditions to penalize controllers that send very low voltages
1038 condVaMin1 = sum(Va1 < VaMinThreshold) >= VaMinRatioThreshold*numel(Va1);
1039 condVaMin2 = sum(Va2 < VaMinThreshold) >= VaMinRatioThreshold*numel(Va2);
1040 condVaMin3 = sum(Va3 < VaMinThreshold) >= VaMinRatioThreshold*numel(Va3);
1041 condVaMin4 = sum(Va4 < VaMinThreshold) >= VaMinRatioThreshold*numel(Va4);
1042
1043 % Conditions to penalize controllers that send very high voltages
1044 condVaMax1 = sum(Va1 > VaMaxThreshold) >= VaMaxRatioThreshold*numel(Va1);
1045 condVaMax2 = sum(Va2 > VaMaxThreshold) >= VaMaxRatioThreshold*numel(Va2);
1046 condVaMax3 = sum(Va3 > VaMaxThreshold) >= VaMaxRatioThreshold*numel(Va3);
1047 condVaMax4 = sum(Va4 > VaMaxThreshold) >= VaMaxRatioThreshold*numel(Va4);
1048
1049 % Compute final condition
1050 if (end) > VaMinThreshold
1051 condVaMin = condVaMin1 || condVaMin2 || condVaMin3 || condVaMin4;
1052 else
1053 condVaMin = false;
1054 end

```

```

1050     if t(end) >= VaMaxTimeThreshold
1051         condVaMax = condVaMax1 || condVaMax2 || condVaMax3 || condVaMax4;
1052     else
1053         condVaMax = false;
1054     end
1055 end
1056
1057 function [condGradPos,condGradNeg] = check_cond_grad(gradErr, signalErrorSigned, ...
1058                                         errThrshld, maxErrThrshld, minErrRatioThrshld)
1059 % Positive gradient
1060 condGradPos = sum(gradErr > 0 & abs(signalErrorSigned) > errThrshld) >= minErrRatioThrshld*numel(gradErr);
1061 condGradPos = condGradPos && signalErrorSigned > maxErrThrshld;
1062 condGradPos = condGradPos && condGradPos;
1063 % Negative gradient
1064 condGradNeg = sum(gradErr < 0 & abs(signalErrorSigned) > errThrshld) >= minErrRatioThrshld*numel(gradErr);
1065 condGradNeg = min(signalErrorSigned) < -maxErrThrshld;
1066 condGradNeg = condGradNeg && condGradNeg;
1067 end
1068
1069 function [rewardTime,rewardZ,rewardPsi] = compute_reward(overallCondGrad, ...
1070                                         overallGradPosZ,condGradNegZ,condGradPosPsi,condGradNegPsi,t)
1071 if overallCondGrad
1072     if t(end) > 0 && t(end) < 3
1073         rewardTime = 0.5;
1074     elseif t(end) >= 3 && t(end) < 6
1075         rewardTime = 0.25;
1076     elseif t(end) >= 6
1077         rewardTime = 0.125;
1078     end
1079     if t(end) > 1 && t(end) < 3
1080         rewardZ = 1;
1081         rewardPsi = 1;
1082     elseif t(end) >= 3 && t(end) < 6
1083         rewardZ = 1.25;
1084         rewardPsi = 1.25;
1085     elseif t(end) >= 6 && t(end) < 8
1086         rewardZ = 1.5;
1087         rewardPsi = 1.5;
1088     elseif t(end) >= 8
1089         rewardZ = 1.75;
1090         rewardPsi = 1.75;
1091     else
1092         rewardZ = 0.5;
1093         rewardPsi = 0.5;
1094     end
1095     if condGradPosZ || condGradNegZ
1096         rewardZ = 0;
1097     end
1098     if condGradPosPsi || condGradNegPsi
1099         rewardPsi = 0;
1100     end
1101 else
1102     if t(end) >= 1 && t(end) < 3
1103         rewardTime = 1.5;
1104         rewardZ = 1.5;
1105         rewardPsi = 1.5;
1106     elseif t(end) >= 3 && t(end) < 6
1107         rewardTime = 2;
1108         rewardZ = 2;
1109         rewardPsi = 2;
1110     elseif t(end) >= 6 && t(end) < 8
1111         rewardTime = 2.5;
1112         rewardZ = 2.5;
1113         rewardPsi = 2.5;
1114     elseif t(end) >= 8
1115         rewardTime = 3;
1116         rewardZ = 3;
1117         rewardPsi = 3;
1118     else
1119         rewardTime = 1;
1120         rewardZ = 1;
1121         rewardPsi = 1;
1122     end
1123 end
1124 end
1125
1126 end

```

E. Build Neural Network

```

1 %% build_neural_network() (Main)
2 function [neuralNet, nbelays, varargout] = build_neural_network(Individual)
3 % This function generate a neural network from each individual's codeGenes and
4 % connectionGenes
5 % References: https://goo.gl/FchG0z
6 % https://goo.gl/rdtEzo
7 % https://goo.gl/odfpvB (genFunction)
8 %
9 % Note: Neural network toolbox uses layers for nodes since a layer can contains
10 % multiple neurons. However, NEAT generates only nodes with single neuron. Hence,
11 % in this function, layers correspond to nodes.
12 %
13 % Also, NEAT's input nodes (nodeType == 1) are input ports in Neural network
14 % toolbox's terminology. The Neural network toolbox's input layers are the
15 % layers directly connected to the input ports. To simplify things, we'll define
16 % NEAT's input nodes as input layers with weight = 1 and linear activation
17 % function. NEAT's output nodes corresponds to the toolbox's last layer before
18 % connection to output.
19 %
20 %% Preprocessing
21 %
22 % Extract node ID and node type
23 nodeID = Individual.nodeGenes(1,:);
24 nodeType = Individual.nodeGenes(2,:);
25 %
26 % Sort nodeID in ascending order
27 [nodeID, sortIndex] = sort(nodeID);
28 %
29 % Sort nodeType according to sortIndex
30 nodeType = nodeType(sortIndex);
31 %
32 % We put output layer at the end for visualisation purposes (when using the view
33 % function)
34 nodeID = [nodeID(nodeType == 2), nodeID(nodeType == 2)];
35 nodeType = [nodeType(nodeType == 2), nodeType(nodeType == 2)];
36 %
37 % Extract connections info
38 connID = Individual.connectionGenes(1,:);
39 connFrom = Individual.connectionGenes(2,:);
40 connTo = Individual.connectionGenes(3,:);
41 connWeight = Individual.connectionGenes(4,:);
42 connEnabled = Individual.connectionGenes(5,:);
43 %
44 % We remove disabled connections
45 indexEnabled = connEnabled == 1;
46 connID = connID(indexEnabled);
47 connFrom = connFrom(indexEnabled);
48 connTo = connTo(indexEnabled);
49 connWeight = connWeight(indexEnabled);
50 connEnabled = connEnabled(indexEnabled);
51 %
52 % Sort connID in ascending order

```

```

53 [connId, sortIndex] = sort(connId);
54
55 % Sort connFrom, connTo, connWeight and connEnabled according to sortIndex
56 connFrom = connFrom(sortIndex);
57 connTo = connTo(sortIndex);
58 connWeight = connWeight(sortIndex);
59 connEnabled = connEnabled(sortIndex);
60
61 % Inputs' node ID
62 inputId = nodeId(nodeType == 1);
63
64 % Outputs' node ID
65 outputId = nodeId(nodeType == 2);
66
67 % Hidden nodes' node ID
68 hiddenId = nodeId(nodeType == 3);
69
70 % Biases' node ID. NEAT uses only one bias for all the networks and choose
71 % whether to connect it to other nodes or not. By assigning a different weight
72 % to every connection from the bias to a node, it is as if these nodes had their
73 % own bias.
74 biasId = nodeId(nodeType == 4);
75
76 %% Network architecture
77
78 % Initialize network object
79 neuralNet = network();
80
81 % Number of nodes
82 nNodes = numel(nodeId);
83
84 % Number of connections (including biases)
85 nConn = numel(connId);
86
87 % Number of biases. NEAT uses only one bias for all the networks and choose
88 % whether to connect it to other nodes or not. By assigning a different weight
89 % to every connection from the bias to a node, it is as if these nodes had their
90 % own bias.
91 nBiases = sum(nodeType == 4);
92
93 % Number of inputs
94 neuralNet.numInputs = sum(nodeType == 1);
95
96 % Number of layers
97 % Neural network toolbox does not consider biases as input nodes
98 neuralNet.numLayers = nNodes - nBiases;
99
100 % Note: Neural network toolbox uses layers for nodes since a layer can contains
101 % multiple neurons. However, NEAT generates only nodes with single neuron. So,
102 % layers correspond to nodes here.
103
104 % Array with first column containing the layer ID and second column its
105 % corresponding ID. We need to do this since neural network toolbox does not
106 % consider biases as input nodes
107 layerList = [1:neuralNet.numLayers]',[nodeId(nodeType == 4)'];
108
109 % Inputs node's layer ID according to layerList
110 inputLayerId = layerList(ismember(layerList(:,2),inputId),1);
111
112 % Outputs node's layer ID according to layerList
113 outputLayerId = layerList(ismember(layerList(:,2),outputId),1);
114
115 % Hidden node's layer ID according to layerList
116 hiddenLayerId = layerList(ismember(layerList(:,2),hiddenId),1);
117
118 %% Bias connections
119
120 % We connect the biases to their layers
121 for jj = biasId
122 % Find nodes connected to this bias. connFrom == jj gives a logical indexing
123 % vector of the node that receive a connection from the bias.
124 % connTo(connFrom == jj) gives the actual node IDs connected to the bias.
125 iNodeConnectedToBias = connTo(connFrom == jj);
126 % Find layers connected to this bias
127 iLayerConnectedToBias = [];
128 for kk = 1:numel(iNodeConnectedToBias)
129 iLayerConnectedToBias = ...
130 [iLayerConnectedToBias, ...
131 layerList(layerList(:,2) == iNodeConnectedToBias(kk),1)];
132 end
133 % Set bias for the above layers (if any)
134 if ~isempty(iLayerConnectedToBias)
135 for ii = iLayerConnectedToBias
136 % neuralNet.biasConnect(T0 ith layer)
137 neuralNet.biasConnect(ii) = 1;
138 end
139 end
140 end
141
142 %% Connect inputs, outputs and hidden layers
143
144 % Input counter for the following loop
145 inputCounter = 1;
146
147 % Cycle through layers (nodes)
148 for iNode = 1:nNodes
149
150 %% Input connections
151
152 % We specify which layers are input layers. Note that this loop assumes that
153 % the input nodes' IDs come before the hidden and output layers.
154 if nodeType(iNode) == 1 % If is input node
155 % Find layer connected to this input
156 iLayerConnectedToInput = layerList(layerList(:,2) == nodeId(iNode),1);
157 % Set input for the above layers (if any)
158 if ~isempty(iLayerConnectedToInput)
159 neuralNet.inputConnect(T0 ith layer, FROM jth input)
160 neuralNet.inputConnect(iLayerConnectedToInput,inputCounter) = 1;
161 end
162 inputCounter = inputCounter + 1;
163 end
164
165 %% Layer connection from input layer and hidden layers
166
167 % We connect input layers to hidden layers as well as hidden layers to other
168 % hidden layers
169 if nodeType(iNode) == 1 || nodeType(iNode) == 2 || nodeType(iNode) == 3 % If is input or hidden node
170 % Find layers that receive connection FROM this node
171 iNodeConnectedFromCurLayer = connTo(connFrom == nodeId(iNode));
172 % Find layers that receive connection FROM this node
173 iLayerConnectedFromCurLayer = [];
174 for kk = 1:numel(iNodeConnectedFromCurLayer)
175 iLayerConnectedFromCurLayer = ...
176 [iLayerConnectedFromCurLayer, ...
177 layerList(layerList(:,2) == iNodeConnectedFromCurLayer(kk),1)];
178 end
179 % Set FROM current layer for the above layers (if any)
180 if ~isempty(iLayerConnectedFromCurLayer)
181 for ii = iLayerConnectedFromCurLayer
182 % neuralNet.layerConnect(T0 ith layer, FROM jth layer)
183 iTp = layerList(layerList(:,2) == nodeId(ii),1);
184 neuralNet.layerConnect(ii,iTp) = 1;
185 end
186 end
187
188 %% Output connections
189
190 % We specify which layers are output layers
191 if nodeType(iNode) == 2 % If is output node
192 % Find layer ID corresponding to current node ID
193 iLayerConnectedToOutput = layerList(layerList(:,2) == nodeId(iNode),1);
194
195 % Set FROM for this output
196 if ~isempty(iLayerConnectedToOutput)
197 % neuralNet.outputConnect(FROM ith layer)
198 neuralNet.outputConnect(iLayerConnectedToOutput) = 1;
199 end
200 end
201
202 end
203
204 %% Set layer connection and delays for recurrent connections
205 % We use MATLAB's graph theory toolbox (https://goo.gl/00cbX5) to find
206 % recurrences in the network
207
208 % First, remove "from connections" of biases from connFrom
209 connFromNoBias = connFrom;
210 connToNoBias = connTo;
211 iBias = 1:iBiases;
212 indexOfBias = connFromNoBias == biasId(iBias);
213 connFromNoBias = connFromNoBias(indexOfBias);
214 connToNoBias = connToNoBias(indexOfBias);
215 end
216
217 % Next, create start and target layers arrays
218 % Important: This will fail if no input are connected!
219 startLayers = [];
220 targetLayers = [];
221 for ii = 1:numel(connFromNoBias)
222 startLayers = [startLayers,layerList(layerList(:,2) == connFromNoBias(ii),1)];
223 targetLayers = [targetLayers,layerList(layerList(:,2) == connToNoBias(ii),1)];
224 end
225 tmpArray = [...];
226 startLayers;
227 targetLayers';
228 tmpArray = unique(tmpArray,'rows');
229 startLayers = tmpArray(:,1)';
230 targetLayers = tmpArray(:,2)';
231
232 %% Build directed graph object
233 directedGraph = digraph(startLayers, targetLayers);
234
235 %% Find recurrent connections
236 recurrentConnection = dfsearch(directedGraph, 1, 'edgetodiscovered', 'Restart', true);
237 %% Plot(directedGraph, 'Layout', 'force');
238
239 %% Find connections that go from one output to another output
240 % excluding output that goes onto itself
241 for ii = 1:numel(connFromNoBias)
242 condition1 = nodeType(nodeId == connFromNoBias(ii)) == 2;
243 condition2 = nodeType(nodeId == connToNoBias(ii)) == 2;
244 condition3 = nodeD(nodeId == connFromNoBias(ii)) == nodeId(nodeId == connToNoBias(ii));
245 if condition1 & condition2 & condition3
246 recurrentConnection = ...
247 [recurrentConnection;
248 [layerList(layerList(:,2) == connFromNoBias(ii),1) == connToNoBias(ii),1]...
249 layerList(layerList(:,2) == connToNoBias(ii),1)...];
250 ];
251 end
252 end
253
254 %% Finally, assign delays to recurrent connections (layers)
255 for ii = 1:numel(recurrentConnection(:,1))
256 % neuralNet.layerWeights(T0 ith layer, FROM jth layer).delays
257 iLayer = recurrentConnection(ii,2);
258 jLayer = recurrentConnection(ii,1);
259 neuralNet.layerConnect(iLayer,jLayer) = 1; % Is this necessary ?
260 neuralNet.layerWeights(iLayer,jLayer).delays = 1;
261 end
262
263 %% Inputs and hidden layers size and activation function
264
265 % Set input sizes (all inputs are size 1)
266 neuralNet.inputs{1}.size = 1;
267
268 % Set layers size, activation function and weight initialization function
269 % We don't use the weight initialization function but the toolbox requires it
270 neuralNet.layers{1}.size = 1;
271 neuralNet.layers(hiddenLayerId).transferFcn = 'logsig';
272 neuralNet.layers(outputLayerId).transferFcn = 'logsig';
273 neuralNet.layers(inputLayerId).transferFcn = 'purelin';
274 neuralNet.layers{1}.initFcn = '';% initnw
275
276 %% Set network functions
277
278 % We don't need those but Neural network toolbox requires them in order to
279 % initialize the network
280 neuralNet.initFcn = ''; % mse
281
282 neuralNet.performFcn = ''; % mse
283 neuralNet.trainFcn = ''; % trainlm
284
285 neuralNet.divideFcn = ''; % dividerand
286
287 % neuralNet.plotFcns = {'plotperform','plottrainstate'};
288 neuralNet.plotFcns = {};
289
290 %% Initialize network
291
292 neuralNet = init(neuralNet);
293
294 %% Set biases weights
295
296 for jj = biasId
297 % Find nodes connected to this bias (i.e. nodes that receive connection from
298 % the bias)
299 iNodeConnectedToBias = connTo(connFrom == jj);
300 % Find layers connected to this bias
301 iLayerConnectedToBias = [];
302 for kk = 1:numel(iNodeConnectedToBias)
303 iLayerConnectedToBias = ...
304 [iLayerConnectedToBias, ...
305 layerList(layerList(:,2) == iNodeConnectedToBias(kk),1)];
306 end
307 % Set bias
308 if ~isempty(iLayerConnectedToBias)
309 for kk = 1:numel(iLayerConnectedToBias)
310 ii = iLayerConnectedToBias(kk);
311 biasWeight = connWeight(connFrom == jj & connTo == iNodeConnectedToBias(kk));
312 neuralNet.b(ii) = biasWeight;
313 end
314 end
315
316
317 %% Set layers weights
318
319 % Input counter for the following loop
320 inputCounter = 1;
321
322 %% Cycle through layers (nodes)
323 for iNode = 1:nNodes
324
325 %% Weight input layer
326
327 if nodeType(iNode) == 1 % If is input node
328 % Find input's layer Id connected to this input
329 iLayerConnectedToInput = layerList(layerList(:,2) == nodeId(iNode),1);
330 % Set weight
331 if ~isempty(iLayerConnectedToInput)
332 % neuralNet.IW(T0 ith layer, FROM jth input)
333 neuralNet.IW(iLayerConnectedToInput,inputCounter) = 1;
334 end
335 inputCounter = inputCounter + 1;
336 end

```

```

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

```

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 separete 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*)\r','tokens');
18   else
19     out1 = {};
20     out2 = regexp(text,'ai(\d*)\r','tokens');
21   end
22   out1 = [out1{:}];
23   out2 = [out2{:}];
24   out = [out1,out2];
25   out = unique(out);
26   recurLayerId = sort(str2double(out));
27 end
28
29 if nPreviousStates >= 1
30   newCodeStart = [];
31   % afMat = zeros\1, num2str(numel(recurLayerId)), '\n';
32   newCodeEnd = ['afMat = zeros\1, num2str(numel(recurLayerId)), \1\n'];
33   % Form code to add in function
34   for ilayer = 1:numel(recurLayerId)
35     % Code to add at the beginning of the file
36     newCodeStart = [...;
37       newCodeStart, ...
38       ['ai\1 num2str(recurLayerId(ilayer)), \1 = aiMat\1, num2str(ilayer), \1\n']];
39     % Code to add at the end of the file
40     newCodeEnd = [...;
41       newCodeEnd, ...
42       ['afMat\1, num2str(ilayer), \1 = af\1, num2str(recurLayerId(ilayer)), \1\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(.*)\r', 'aiMat\r');
51 else
52   find_and_replace(funPath,[funName,'(\.\.\.\r)',[funName,'(\$\1,aiMat)\r']]);
53 end
54
55 % Replace af1,af2,etc. by afMat
56 if nPreviousStates > 1
57   find_and_replace(funPath,'af(.*)\r', 'afMat\r');
58 else
59   find_and_replace(funPath,['\(\.\.\.\r)',funName,['\$\1,afMat\r',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',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(-a\r)', 'exp(-4.9*a\r)');
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.pdf (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).