We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

28

Modelling and Identification of Flight Dynamics in Mini-Helicopters Using Neural Networks

Rodrigo San Martin Muñoz, Claudio Rossi and Antonio Barrientos Cruz
Universidad Politécnica de Madrid, Robotics and Cybernetics Research Group
Spain

1. Introduction

Unmanned Aerial Vehicles have widely demonstrated their utility in military applications. Different vehicle types - airplanes in particular - have been used for surveillance and reconnaissance missions. Civil use of UAVs, as applied to early alert, inspection and aerial-imagery systems, among others, is more recent (OSD, 2005). For many of these applications, the most suitable vehicle is the helicopter because it offers a good balance between manoeuvrability and speed, as well as for its hovering capability.

A mathematical model of a helicopter's flight dynamics is critical for the development of controllers that enable autonomous flight. Control strategies are first tested within simulators where an accurate identification process guarantees good performance under real conditions. The model, used as a simulator, may also be an excellent output predictor for cases in which data cannot be collected by the embedded system due to malfunction (e.g. transmission delay or lack of signal). With this technology, more robust fail-safe modes are possible.

The state of a helicopter is described by its attitude and position and the characteristics of its dynamics system correspond to those of a non-linear, multivariable, highly coupled and unstable system (Lopez, 1993). The identification process can be performed in different ways, on analytical, empirical or hybrid models, each with its advantages and disadvantages.

This Chapter describes how to model the dynamic of a mini-helicopter using different kinds of supervised neural networks, an empirical model. Specifically, the networks are used for the identification of both attitude and position of a radio controlled mini helicopter. Different hybrid supervised neural network architectures, as well as different training strategies, will be discussed and compared on different flight stages. The final aim of the identification process is to build a realistic flight model to be incorporated in a flight simulator.

Although several neural network-based controllers for UAVs can be found in the literature, there is little work on flight simulator models. Simulators are valuable tools for in-lab testing and experimenting of different control algorithms and techniques for autonomous flight. A model of a helicopter's flight dynamics is critical for the development of good a simulator. Moreover, a model may also be used during flight as predictor for anticipating the behaviour of the helicopter in response to control inputs.

The Chapter first focuses on two neural-network architectures that are well suited for the particular case of mini-helicopters, and describes two algorithms for the training of such neural-network models. These architectures can be used for both multi-layer and radial-based

hybrid networks. The advantages and disadvantages of using neural networks will also be discussed. Then, a methodology for acquiring the training patterns and training the networks for different flight stages is presented, and an algorithm for using the networks during simulations is described. The methodology is result of several years of experience in UAVs. Finally, the two architectures and training methods are tested on real flight data and simulation data, and the results are compared and analysed.

2. Network Architectures for Modelling Dynamics Systems

Modelling a dynamic system like a mini-helicopter, requires estimating the effect of both the inputs and the system's internal state on the outputs (Norgaard et al., 2001). Considering the system's identification by means of state variables, a dynamic system can be described in a discrete space as shown in (1), where v is the state variable, $\boldsymbol{x}$ the input and $\boldsymbol{y}$ the output.

$$\begin{aligned} v(k+1) &= \Phi[v(k), x(k)] \\ y(k) &= \Psi[v(k), x(k)] \end{aligned} \tag{1}$$

The first equation shows the dependence of the state at a certain time with regard to the state and inputs in a previous instant, similar to recurrent neural networks, for example Hopfield Neural Network (Hopfield, 1982). The second equation shows the dependence of the outputs in instant k with regard to the state and inputs in the same instant, as in non-recurrent neural networks, for example Radial Basis (RB) or multi-layer Perceptron (MLP) Neural Network (Freeman & Skapura, 1991). These equations suggest the use of a mixed neural network with recurrent and non-recurrent properties (Narendra & Parthasarathy, 1990). The recurrent component is used to describe the system's state, while the non-recurrent component defines the system's outputs. There are two mains types of mixed networks, Jordan's (Jordan, 1986) and Elman's (Elman, 1990). In this case, theses networks are not applicable. For this reason, will be used a new proposed hybrid networks which is suitable for modelling of flight dynamics in mini-helicopters.

2.1 Jordan's Network

Jordan's network (Jordan, 1986) consists of a multi-layer network with external inputs $\mathbf{X=[x_1\ x_2\ ...\ x_n]}$ and contextual neurons $\mathbf{C=[c_1\ c_2\ ...\ c_m]}$, that represent the internal states (see Figure 1). These contextual neurons are recurrent because they use both their previous output $\mathbf{C(t-1)}$ and the previous system's output as inputs. This means that they store the systems' past states adjusted by μ (store weight), as shown in (2).

This architecture works, basically, as a multi-layer network with the particularity that the network's external input and contextual neurons create a new input vector $\mathbf{U=[x_1\ x_2\ ...\ x_n\ c_1\ c_2\ ...\ c_m]}$. In (2) it is possible to observe that the i-th output corresponds to the composition of the outputs of every layer, as in a non-recurrent network.

$$\begin{aligned} C_i(k) &= \mu \cdot C_i(k-1) + y_i(k-1) \forall i = 1,2,...,m \\ C_i(k) &= \sum_{j}^{k-1} (\mu^{j-1} y_i(k-1)) \forall i = 1,2,...,m \\ y_i(k) &= S_i(\sum_{j=1}^{j=g} N_j(\sum_{h=1}^{n+m} u_h \cdot w_{jh}) \cdot w_{ij}) \forall i = 1,2,...,m \end{aligned} \tag{2}$$

Where u_i corresponds to the elements of vector **U**, $\mathbf{w_{jh}}$ are the weights in the hidden layers (**N**), $\mathbf{w_{ij}}$ are the weights in the output layer (**S**) and μ is the weight of time constant.

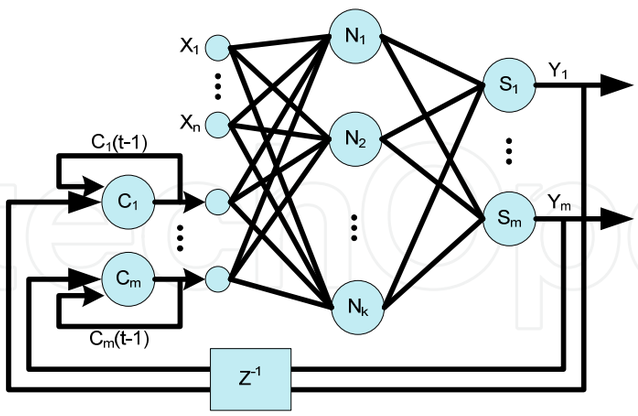


Figure 1. Jordan's network, where Z^{-1} is a sample delayed in the time

Rewriting (2) results in (3), where **Y** it is defined by a function whose inputs are the elements u, which are defined by vectors **C** and **X**, and **C** is defined by a weighted sum of **Y,** which is nothing more than **C** and **X**.

$$\begin{aligned} C(k+1) &= F[C(k), X(k)] \\ Y(k) &= G[C(k), X(k)] \end{aligned} \tag{3}$$

2.2 Elman's Network

Elman's network (Elman, 1990) (Fig. 2), unlike Jordan's, does not feed back the contextual neuron output or the system output, as shown in (4). Instead, it feeds back the output of an intermediate layer.

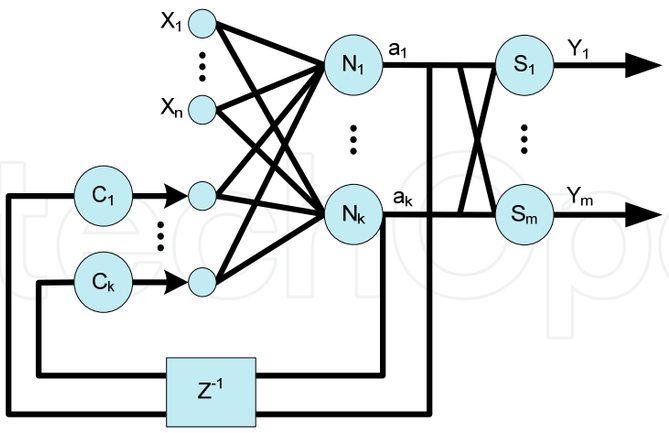


Figure 2. Elman's network

The input vector is exactly the same as in Jordan's network, the only difference being the value of the contextual neurons, as defined in (4). In this case, the contextual neurons do not preserve past states, but save only the last state value.

$$C_i(k) = a_i(k-1) \forall i = 1,2,...,k \tag{4}$$

2.3 Proposed Hybrid Network

Other neural network architectures are possible, in which the context neurons appear only in the first layer of the network. The architecture of (Narendra & Parthasarathy, 1990) tries to generate the contextual neurons at all levels, that is, both for the outputs and for the inputs. This idea is the basis of the architecture that will be developed in this work.

The proposed hybrid network consists of two blocks: the first is the recurrent component with context neurons for the inputs and outputs. These neurons use past states as inputs and the number of states is defined automatically by the training algorithm or by means of some stochastic model. The second block is a non-recurrent network that may be either an MLP or an RB. Fig. 2.3 shows a hybrid network in which Block A consists of a non-recurrent network and Block B corresponds to context neurons (recurrent network). Block's B neurons do not follow Elman's or Jordan's architecture, but rather a mix: the feedback is **d** past system outputs and **h** past system inputs. The number of past states to use is adjustable, as is the number of neurons in the hidden layer of Block A. This flexibility is necessary because the system is unknown and the network must adapt during the training process to attain the minimum error.

The external inputs are represented by vector $\mathbf{X=[x_1\ x_2\ ...\ x_n]}$ and these are stored in the contextual neurons $\mathbf{C_{xi1}}$ to $\mathbf{C_{xid}}$. The outputs are represented by vector $\mathbf{Y=[y_1\ y_2\ ...\ y_m]}$ which is also stored in contextual neurons $\mathbf{C_{yi1}}$ to $\mathbf{C_{yih}}$. As opposed to Jordan's network (2) these contextual neurons keep **d** past inputs and **h** past outputs, which yields the vectors $\mathbf{C_{xi}=[x_i(k\text{-}1)\ \dots\ x_i(k\text{-}d)]}$ and $\mathbf{C_{yi}=[y_i(k\text{-}1)\ \dots\ y_i(k\text{-}h)]}$ for the i-th iteration. Thus, the input vector for the non-recurrent network in Block A is $\mathbf{U=[X\ C_{x1}\ \dots\ C_{xn}\ C_{y1}\ C_{ym}]}$. This means that **U** contains **n** elements from the input vector **X**, **n** elements for every previous state **d**, **m** elements from the output vector **Y** and **m** elements for every previous state **h**. To sum up, the amount of elements for vector **U** is: $\mathbf{n+(n \bullet d)+(m \bullet h)}$.

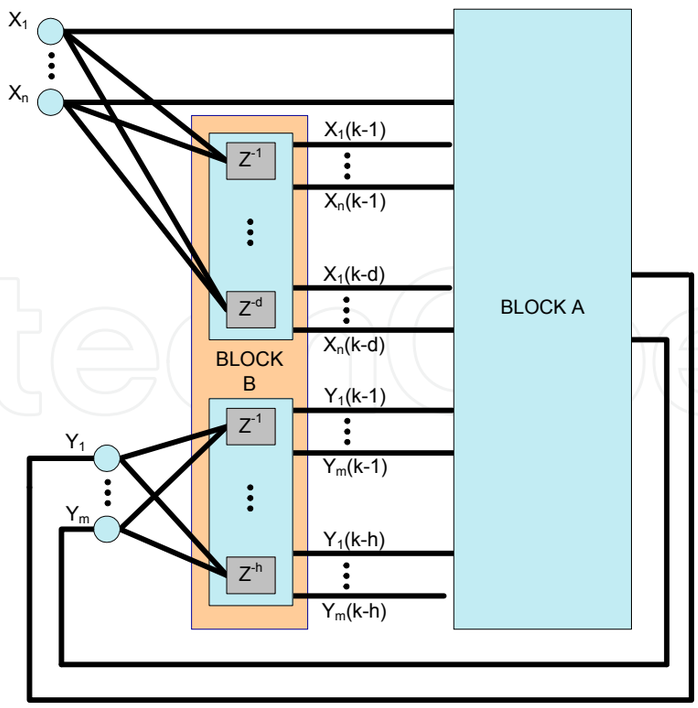


Figure 3. Hybrid network

A slightly modified Elman's network is very close to this idea, the main difference being that Elman's feeds back from the hidden layer. The modification consists of a feedback loop that involves not only a delay $\mathbf{Z^{-1}}$, but also a delay block similar to **Block B** in the hybrid network (Fig. 3). On the other hand, the Jordan's network does feed back the output, but does not consider all previous states with the same weight. Equation (2) shows how each previous value is stored in the memory of the contextual neuron itself, which causes the value of the contextual neuron to be the result of adding all the weighted previous states. In other words, the hybrid network has a finite number of previous states that generate the same amount of contextual neurons; in contrast, Jordan's network generates a contextual neuron with an infinite amount of previous states by performing a weighted sum of them.
When creating a hybrid network, principles from both networks are taken into account, but the possibility of additional modifications for future improvement is left open. For example, consider a contextual neuron for storing the rest of the previous states, as done by Jordan. Also, it is possible to feed back the output of the first layer, as suggested by Elman. The possibilities are manifold, but it is necessary to choose one architecture in order to be able to start performing any tests.
As mentioned before, **Block B** corresponds to the recurrent stage, where the previous states are stored, and it is an integral part of the network architecture. In the case of a system with substantial inertia, the order of the delays (**d** and **h**) will increase. **Block A** corresponds to the non-recurrent stage, which performs the system output signal tracking and is able to operate internally both with Multi-layer Perceptron and Radial Basis networks.
Similar to the mixed networks mentioned previously, the hybrid network can be converted to the form (3), where functions **F** and **G** will depend on the architecture selected for **Block A**, either an RB or MLP network, as well as the internal transfer functions of the recurrent networks. The following section describes the hybrid network used for the UAV system and justifies its architecture.

3. Proposed Hybrid Network Architecture for the Identification of a Mini-Helicopters (UAV)

For the identification of a system like the mini-helicopter, some adjustments need to be made to the hybrid network and the simulation strategy must be planned. A helicopter flight is based on the angle-of-attack and angular velocity of its blades. These values define the attitude and lift that, in turn, change the position of the aircraft (Lopez, 1993). Due to these circumstances, two stages are considered in the identification process: computing the attitude using the control commands as inputs, and then using the attitude to obtain, in this case, the vehicle position. Hybrid networks can be used to model both stages (Fig. 2.3).
In summary, the dynamic system to be modelled corresponds to a system which, after receiving some control commands, modifies its attitude ($\boldsymbol{\theta, \phi, \psi}$) and consequently, its position (**X, Y, Z**) (as shown in Fig. 4).
The neural network's outputs are the helicopter's attitude and position, and its inputs are the roll, pitch and yaw cyclic steps and the collective, labelled Croll, Cpitch, Cyaw and Ccole, respectively. Croll and Cpitch control the cyclic angle-of-attack of the main rotor blades, Cyaw commands the tail rotor and Ccole is a combination of the main rotor blade's angle-of-attack and the engine throttle. These control signals are the same ones a human pilot uses to command a mini-helicopter and represent the inputs of the radio-controller.

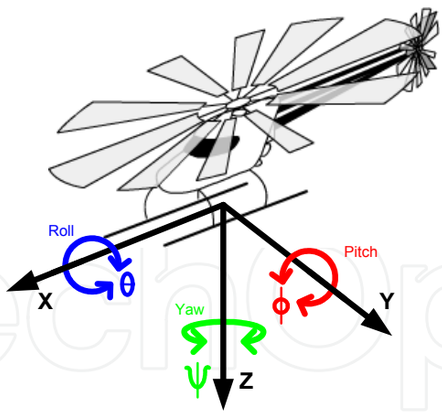


Figure 4. Attitude angle and position

With this architecture, based on two hybrid networks, two training methods are possible. The first connects the system as a daisy chain: the output of the attitude system's training is used as the input for the position system's training. The second training method places the systems in parallel and uses real flight data to train both networks. It is important to note that both training methods require carefully selected parameters: number of inputs n and outputs m, order of contextual neurons for inputs d and outputs h and type of network in Block A (MLP or RB). The method used to set these parameters will be described in the following sections.

3.1 Training Architectures

The data obtained from the avionics (attitude: θ roll, ϕ pitch, ψ yaw, and position: X, Y, Z) and the radio transmitter (control commands: Croll, Cpitch, Cyaw, Cole), are the patterns used for the training. The position and attitude degrees of a helicopter's flight system are depicted in Fig.4, where its position being defined by its attitude.

As mentioned above, there are two training methods:

Daisy chain Architecture: the attitude system is trained as a single and isolated system, which is possible thanks to the previous knowledge of the attitude data (roll θ, pitch ϕ, yaw ψ). The values obtained from the attitude network, estimated data, (roll', pitch' and yaw') are used as inputs for training the position network.

For the attitude system, the training pattern for **Block A_a** is vector $\mathbf{P_a}$ (5), with an external input vector $\mathbf{X_a}$ consisting of the different radio control commands (Croll, Cpitch, Cyaw and Ccole), another vector name $\mathbf{C_{in\_a}}$, that corresponds to the input contextual neurons, and finally $\mathbf{C_{out\_a}}$, which represents the attitude system's feedback output contextual neurons. The training pattern is represented by vector $\mathbf{T_a}$ (6), which is the real attitude data provided by the avionics.

$$\begin{aligned} P_a &= [X_a, C_{in\_a}, C_{out\_a}] \\ X_a &= [Croll(t), Cpitch(t), Cyaw(t), Cole(t)] \\ C_{in\_a} &= [C_{Croll}, C_{Cpitch}, C_{Cyaw}, C_{Cole}] \\ C_{out\_a} &= [C_{roll}, C_{pitch}, C_{yaw}] \\ C_i &= [i(t-1),..., i(t-d)] \forall i = Croll, Cpitch, Cyaw, Cole \\ C_j &= [j(t-1),..., j(t-h)] \forall i = roll, pitch, yaw \end{aligned} \tag{5}$$

$$T_a = [roll\,(t),\ pitch\,(t),\ yaw\,(t)] \tag{6}$$

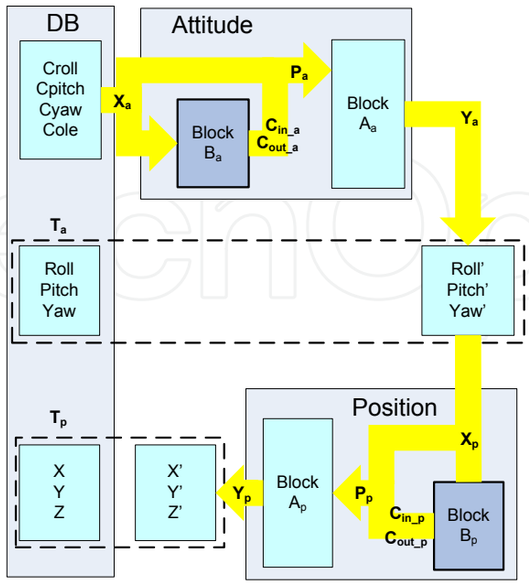


Figure 5. Daisy chain Architecture

Once the patterns are obtained, the training of **Block $\mathbf{A_a}$** starts by comparing the desired system output $\mathbf{T_a}$ (6) with the current network output $\mathbf{Y_a}$ (7).

$$Y_a = [roll',\ pitch',\ yaw'] \tag{7}$$

All necessary adjustments are performed with this *error* according to the training rules of a recurrent network. Basically, as long as the network is correctly trained, a minimum error is expected in the comparison between the desired output $\mathbf{T_a}$ and the real output $\mathbf{Y_a}$.
After adequate training of the attitude, the network is used as a simulator and its $\mathbf{Y_a}$ vector (7) is used as the input pattern for the position system. This pattern will be very similar to the one used in the previous network, the main difference being the input vector $\mathbf{X_p}$ (8), which does not have avionics-acquired values but simulated data from the previous network. The output pattern for the training is vector $\mathbf{T_p}$ (9), which contains the avionics-acquired (GPS) position.

$$\begin{aligned} P_p &= [X_p, C_{in\_p}, C_{out\_p}] \\ X_p &= [roll'(t), pitch'(t), yaw'(t)] \\ C_{in\_p} &= [C_{roll'}, C_{pitch'}, C_{yaw'}] \\ C_{out\_p} &= [C_X, C_Y, C_Z] \\ C_i &= [i(t-1),..., i(t-d)] \forall i = roll', pitch', yaw' \\ C_j &= [j(t-1),..., j(t-h)] \forall i = x, y, z \end{aligned} \tag{8}$$

$$T_p = [x(t),\ y(t),\ z(t)] \tag{9}$$

The value obtained at the output of the network is $\mathbf{Y_p}$ (10)

$$Y_p = [x'(t),\ y'(t),\ z'(t)] \tag{10}$$

Decoupled Training Architecture: the only difference between the training architectures (Fig. 6.) is the input data used for the training process of the position network: a decoupled trainer uses the external input $\mathbf{X_p}$ (11), which contains avionics-acquired attitude values instead of simulated data. The attitude network training is identical.

$$\begin{aligned} P_p &= [X_p, C_{in\_p}, C_{out\_p}] \\ X_p &= [roll(t), pitch(t), yaw(t)] \\ C_{in\_p} &= [C_{roll}, C_{pitch}, C_{yaw}] \\ C_{out\_p} &= [C_X, C_Y, C_Z] \\ C_i &= [i(t-1),..., i(t-d)] \forall i = roll, pitch, yaw \\ C_j &= [j(t-1),..., j(t-h)] \forall i = x, y, z \end{aligned} \tag{11}$$

The process of training both networks (attitude and position) is, in this case, independent, and is done in parallel, because the attitude network outputs are not used for the position network training.

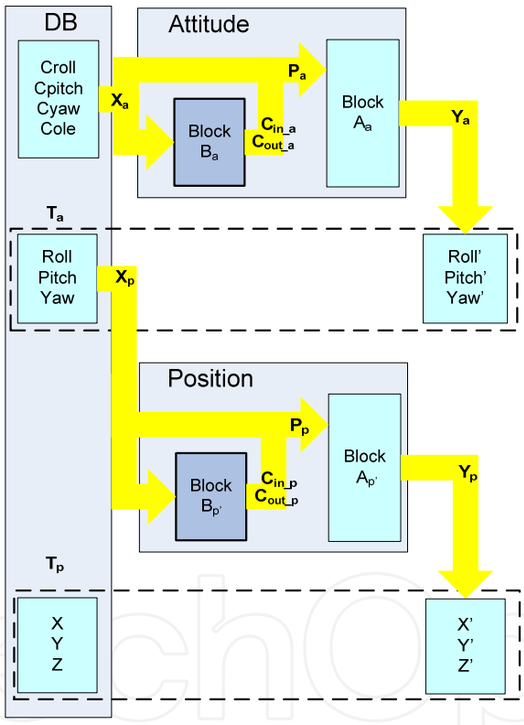


Figure 6. Decoupled architecture

The **training** and **simulation** errors of the attitude network are the same for both architectures, as the training process is the same. The **training** error for the position network is lower when the decoupled architecture is used, and one could assume that this would lead to a lower **simulation** error (when real-time data from the avionics is used to simulate the UAV's position). However, this is not the case. Real-time simulation of the UAV works in daisy chain: flight-data is fed to the attitude network, which in turn feeds the position network. Position networks trained with the daisy-chain architecture have a lower **simulation** error because they have been trained with simulated data and have *learned* to compensate for the simulation error.

3.2 Proposed Hybrid Network Architecture

The modelled system has significant inertia and dynamics. Therefore, the order of the contextual neurons depends on the correlation between the command signals and attitude, and between attitude and position. This correlation is expressed as the delay between a significant change in the inputs and a significant change in the outputs (i.e., inertia). Careful analysis of flight data shows that the delay fluctuates between 500 and 900 ms (the sampling period is 100 ms). Considering worst-case scenarios, a delay of 10 samples was used for the output contextual neurons $\mathbf{C_{out}}$ and a delay of 5 samples was used for the input contextual neurons $\mathbf{C_{in}}$, for both the attitude and the position system. This decision is based on extensive tests that have proven that these values provide the best performance.

The contextual neuron order affects the training patterns and their conformation as well as the number of neurons in the hidden layer of the MLP, which is based on the amount of inputs (Li et al., 1988). The input vector for the attitude network of **Block $\mathbf{A_a}$** is formed as follows: 4 direct inputs from the radio transmitter $\mathbf{X_a}$ (5), which also generates vector $\mathbf{C_{in\_a}}$ (5) with an order $\mathbf{d}$ = 5, for a total of 20 contextual neurons; the output vector $\mathbf{Y_a}$ (7) is used to create vector $\mathbf{C_{out\_a}}$ (5) with an order $\mathbf{h}$ = 10, which results in 30 neurons. All of them yield the input vector $\mathbf{P_a}$ (5), with 54 neurons. This number is kept fixed both for the Radial Basis networks and the MLP. The input vector $\mathbf{P_p}$ (11) for the position system of **Block $\mathbf{A_p}$** has 48 neurons: 3 direct inputs, corresponding to the attitude outputs $\mathbf{X_p}$ (11) that in turn generate the 15 contextual neurons of vector $\mathbf{C_{in\_p}}$, with order $\mathbf{d}$ = 5; the contextual neurons for the output is vector $\mathbf{C_{out\_p}}$ (11), with order $\mathbf{h}$ = 10 and 30 elements.

For both the MLP and the RB the training error threshold is 10^{-5}. The number of epochs for the RB networks is subject to the number of neurons in the hidden layer and is, therefore, variable. For the MLP the number of epochs is fixed at 40,000.

4. Training Pattern Generation

The success of systems identification depends on a good experimental method, even more so with a model based on neural networks. Moreover, it is well known that a wide range of flight scenarios is needed to successfully train the network. This is why it is important to choose the flight data carefully and assess its quality. The data-acquisition equipment is described below, and its capabilities are known. However, there are other important factors to be considered, for example: sampling intervals, wind speed, GPS precision variations, hardware malfunction, vibration, air temperature, etc. For all these reasons many data-gathering flights are needed to guarantee representative and high-quality samples for different kind of conditions and actions (take-off, hover, etc.).

The modelled UAV is an in-house prototype with a 5 kg payload capacity and embedded avionics and control systems, built with a 26cc Benzin Trainer radio-controlled helicopter by Vario. This UAV was developed within Project DPI 2003-01767 of the *Ministerio de Educación y Ciencia of Spain*.

The dynamic system to be modelled corresponds to a system which, after receiving some control commands, modifies its attitude ($\boldsymbol{\theta, \phi, \psi}$) and consequently, its position ($\mathbf{X, Y, Z}$) (as shown in Fig. 3.1). Both the attitude and position are acquired and pre-processed by the avionics with three sensors: a Mircroinfinity A3350M IMU (referential unit), a Honeywell HMR3000 compass and a Novatel OEM-4 DPGS (capable of using RT-2 corrections for 2 cm accuracy). The avionics system is built around a PC-104 board, the OctagonPC/770 with a 1GHz Pentium III processor, running Redhat Linux 8.0 with a Linux/RT kernel. Power is

supplied by an HPWR104+HR DC/DC converter and two 4LP055080+pc 14,8V-2000mAh battery packs, offering two hours of autonomy.
The system's output (attitude, position and related linear/angular velocities and accelerations) and input (control commands) signals are stored synchronously in a data file. Additional information is appended (e.g. GPS signal quality, servo PWM input, etc.) such that different flight phases and actions (take-off, landing, etc.) can be identified and used to build different training patterns for the neural network (Nguyen & Prasad, 1999).

4.1 Acquisition Procedure

The experiments are performed with a helicopter controlled by means of a radio controller in the hands of an expert pilot who commands the vehicle through a set of predefined manoeuvres.
A validation procedure has been established, repeated for each flight, that is, essentially, the first quality filter for the flight data. This procedure, shown in Fig. 7, consists of a permanent *health* evaluation of the helicopter's hardware and software. Usually the hardware (Fig. 7.a) is verified in the lab with routine tests and benchmarks; batteries are fully charged for each test and periodically tested. After a successful boot of the avionics computer, the communication links between the helicopter and the ground station are verified and the DGPS quality is asserted (Fig. 7.b). Then the pilot does an extensive pre-flight verification (e.g. radio-controller range, servo condition, etc.). If all the requirements are met, the system is ready for take-off. The main objective of these tests is to guarantee flawless operation of the helicopter.

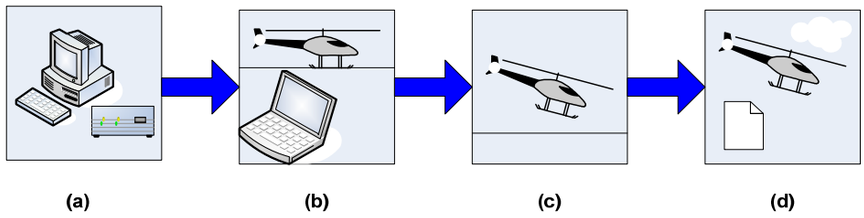


Figure 7. Data acquisition procedure (a) Hardware checking in the lab (b) Ground check of the communications and position systems (c) low height flight (d) data acquisition

A low-height flight ensures that all subsystems are operating correctly and that atmospheric conditions are within pre-established limits (Fig. 7.c). As part of routine checks or when hardware/software malfunction is suspected, the flight data is stored for detailed examination (Fig. 7.c). These tests may reveal subtle problems, such as the intermittent loss of the radio link with the ground station.
Once the system and environmental conditions are considered satisfactory, the system is set up to obtain the experimental data (Fig. 7.d) based on the flight plan. This plan includes five flight stages: start, take-off, manoeuvre, landing and end (see Fig. 8).

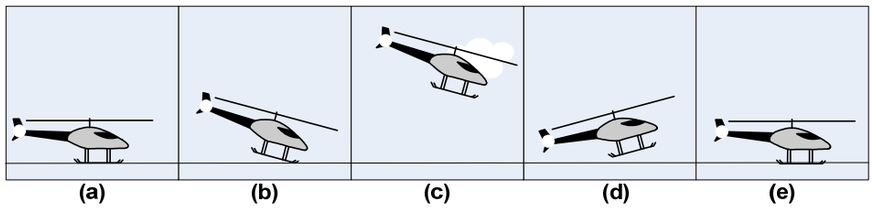


Figure 8. Flight stages stored in the text file. (a) start (b) take-off (c) flight (d) landing (e) end

Start stage: data from the helicopter standing on the ground, initial conditions. (Motor state: off)
Take-off: data from the moment that the helicopter is standing on the ground until it reaches the cruise height, before performing any manoeuvre. These values are affected by the ground effect.
Flight: data from the manoeuvres chosen for the current session (the manoeuvre plan).
Landing: data from the moment the landing procedure starts until the helicopter stands on the ground and stops.
End stage: data after landing; this data and the start data are necessary to check the correct operation of the equipment. (Motor state: off)

4.2 Data Selection Criteria

Flight data is analysed after the data acquisition process. The purpose of these inspections is to validate data quality before the data is used to build the training patterns. Two sets of criteria are established: the first is signal quality, altered by environmental conditions and equipment state. The second set is form: the type of flight performed and the similarity between the desired flight-manoeuvre plan and the actual flight.
Quality criteria: the objective here is to separate samples into suitable and unsuitable. It is important to note that suitability is defined by the requirements of different tasks. For example, samples may be suitable for simulation or training or observation, depending on their quality and significance. The quality criteria are:

- **Atmospheric:** represents the reliability of the data depending on the weather conditions present during the acquisition process, e.g., wind speed.
- **Position data quality:** in general the quality of the GPS solution for position must be better than narrow float (Novatel, 2002).
- **Attitude data quality:** in order to ensure the reliability of this data set, the attitude data obtained from the IMU at the start and end stages is compared (Fig. 8.a and Fig. 4.2.e). Considering a flat surface for the take-off and the landing manoeuvres, roll and pitch must be similar and close to zero.
- **Timing quality:** this criterion verifies the periodicity of the samples (i.e. timestamps). The sampling period is 100 ms and various malfunction conditions may lead to significant deviations. Data with a sampling period of more than 200 ms is considered to be of low quality.

Any sample that does not satisfy all quality criteria is marked unsuitable for training and discarded immediately.
Form criteria: these criteria define the experiment or type of flight. Not all flights are aimed at data acquisition since there are test and training flights (see Fig. 7 b and c). There are three flight types:

- **Standardisation:** corresponds to tests that bring the equipment to its ranges limits, e.g. signal limits for sensors or radio controller. In most cases, these tests are carried out while on the ground.
- **Test Flight:** used for system analysis. The data is not stored for further training, simulations or standardisations; it is only used to correct and measure acquisition errors like, for example, atmospheric conditions.

- **Displacements and Hovering Flight:** corresponds to lateral, longitudinal and vertical displacements and hovering. Different manoeuvres make it possible to train the network under different conditions and scenarios.

4.3 Pattern Transformation

System identification with neural networks requires pre-processing the flight data: normalization, periodicity and yaw adjustments.

Normalisation: in general transfer functions of neural networks operate in the ranges [-1 1] or [0 1]. Therefore, the pattern data must be normalised. Equations (12) and (13), respectively, normalise each entry x_k of vector $\mathbf{X=[x_1 \ldots x_k \ldots x_n]}$ using the maximum and minimum values of X.

$$x_{k\,norm} = \frac{2 \cdot (x_k - \min(X))}{(\max(X) - \min(X))} - 1 \tag{12}$$

$$x_{k\,norm} = \frac{(x_k - \min(X))}{(\max(X) - \min(X))} \tag{13}$$

Periodicity: training a network is considered a process in discrete time. Therefore, it is necessary that samples be obtained periodically. Due to malfunction, the sampling period may not be constant, and depending on the severity of the deviation, samples may be discarded (timing quality factor) or interpolation/extrapolation algorithms may be applied.

Yaw reference: roll and pitch are easily validated since the helicopter's attitude while standing on a horizontal surface must be very close to zero. The yaw reference is the magnetic north and does not necessarily begin or end at zero. To simplify the network training, the initial yaw is considered as an offset so the initial values are close to zero.

5. Hybrid Network Algorithm Description

The training and simulation algorithms were developed with MATLAB, using the Neural Network Toolbox (Demuth & Beale, 2004) and many custom tools that had to be developed since standard toolboxes are not suited to our particular architecture and dynamic characteristics of the modelled system.

5.1 Training Algorithm

Fig. 9 shows the training block for a hybrid network used MLP architecture. Although the training block of a RB network is similar (Freeman & Skapura, 1991), the propagation is different. Pattern **P** is obtained from the data adaptation algorithm and is used for training the hybrid network. The first samples of the state variables **P** are considered as the system's initial values and then the input values are propagated - or their equivalent in the case of the RB network. The system output **Y**, which is calculated for that input sample, enables the calculation of the corresponding errors between **Y** and the train pattern **T**, which are stored in a delta error (**Ed**), however the network's weights are left unchanged. Then the next sample is obtained and the state variables are replaced with the output **Y** calculated previously and result in a new propagation. The cycle continues until the end of the epoch (**i=nsample**). Once the epoch reaches its end, the training error **E** is calculated, the

parameters adjusted (weights and bias) and the training parameters are updated (momentum and learn rate).

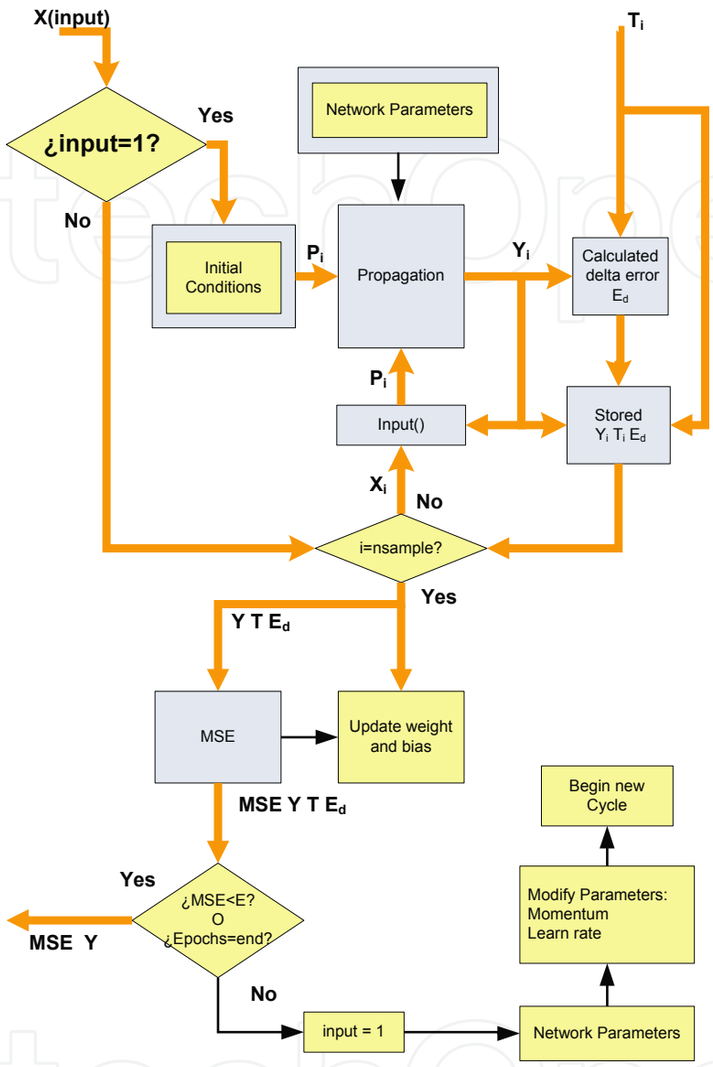


Figure 9. Training Algorithm for Hybrid Network

The process is interrupted when the target error ***E*** is attained or the maximum number of epochs (***end***) is reached. The training process is considered successful only if ***Error*** is lower than **E**, and trainings that reach ***end*** epochs are re-run with adjusted parameters (momentum, learning factor, number of neurons for the hidden layer, etc.).

The specific training algorithms for the decoupled and daisy chain architectures, both for MLP and RB networks, are adaptations of this generic description, defined in Section 3.1.

5.2 Simulation Algorithm

Figure 10 shows the decoupled simulation algorithm for a generic network, which can be applied to attitude or position networks (the only difference is the input vector **X**). The algorithm will run until the final condition is reached (***while(input exists)***), which is the

same as saying that it will run until it iterates through all the input data. The algorithm is capable of running with real-time input data.

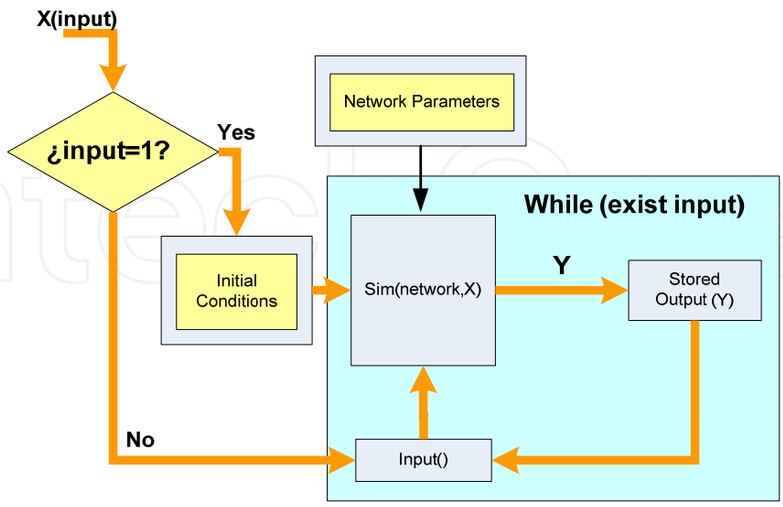


Figure 10. Simulation Algorithm for hybrid network

The first sample (***input=1***) sets the initial conditions, which are determined empirically when running in real-time with real-flight data or obtained from the input file when running with stored flight data. After this initialisation step, the system iterates through the simulation process (***sim(network,input)***) and the output vector **Y** is stored in a matrix, to be used as input in successive iterations.

6. Test and Results

The training patterns were built with data from 9 flight sessions that adhere to the quality and form criteria. Each session has approximately 3 minutes of high-quality flight data. The system identification process will be used to model complete flights and the 5 flight stages (see Fig. 8). The objective is to have different models for each type of manoeuvre and compare the performance of the training architectures and network types (MLP vs. RB).

6.1 Complete Flight vs. Flight stages

The dynamic behaviour of the helicopter is different for each one of the flight stages described in Section 4. For example, the ground effect is present during take-off and landing procedures but is nonexistent above a certain altitude. This is why it is necessary to differentiate between flight stages and to analyse the performance of *universal* models vs. groups of models specialised in different stages.

Fig. 11 shows the results for attitude simulation with MLP (Fig. 11.a) and RB (Fig. 11.b) based networks, both for complete-flight and flight-stage models. Table 1 compares the performance of complete-flight models with the average performance of flight-stage models for three stages (take-off, manoeuvres, landing). Finally, Table 2 shows the mean square error (MSE) for the flight-stage models whose average appears in Table 1.

Table 1 shows that the differences in performance observed between complete-flight and flight-stage models for the attitude simulation are significant. Thus, this experiment has not been repeated for the position simulation since the attitude errors will be propagate.

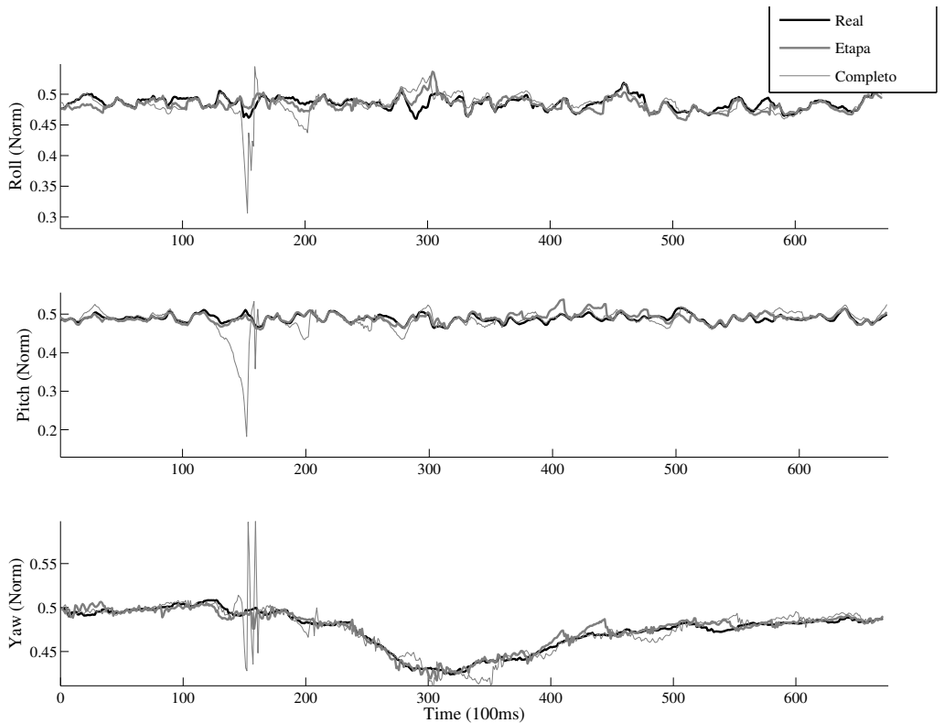


Figure11.a Attitude simulation with MLP

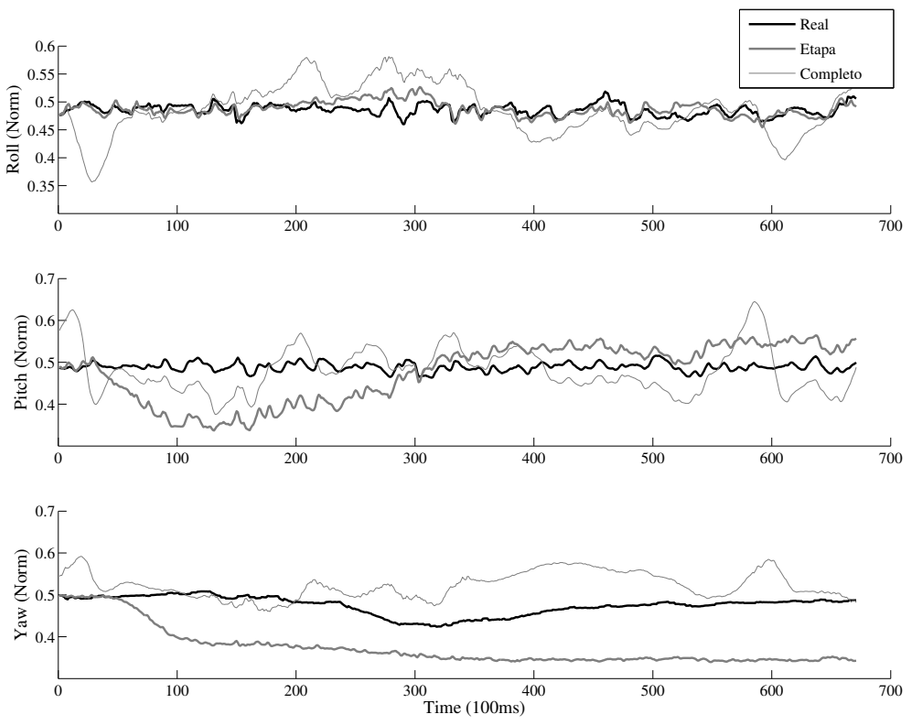


Figure 11.b Attitude simulation with RB

	Complete-flight	Average flight-stages
Attitude	MLP	MLP
Roll	2,27E-04	4,17E-06
Pitch	7,48E-04	2,82E-06
Yaw	1,08E-04	3,51E-06
	RB	RB
Roll	4,56E-03	3,47E-07
Pitch	2,44E-03	3,00E-07
Yaw	5,91E-03	4,23E-07

Table 1. Comparison MSE for complete-flight models with flight-stage models

	Take-off	Manoeuvres	Landing
Attitude	MLP	MLP	MLP
Roll	4,17E-6	8,53E-5	3,13E-6
Pitch	2,82E-6	9,57E-5	4,76E-6
Yaw	3,51E-6	2,63E-5	2,81E-6
	RB	RB	RB
Roll	3,47E-7	1,72E-4	2,34E-6
Pitch	3,00E-7	5,01E-3	2,27E-6
Yaw	4,23E-7	1,18E-2	6,14E-7

Table 2. MSE flight-stage models (take-off, manoeuvres and landing)

6.2 Radial Basis vs. Multi-layer Perceptron

Radial Basis and MLP networks were compared for the same flight stage. Fig. 12.a shows the attitude simulation for both networks versus the real attitude. Fig. 12.b shows the position simulation for each network during take-off versus the real position.

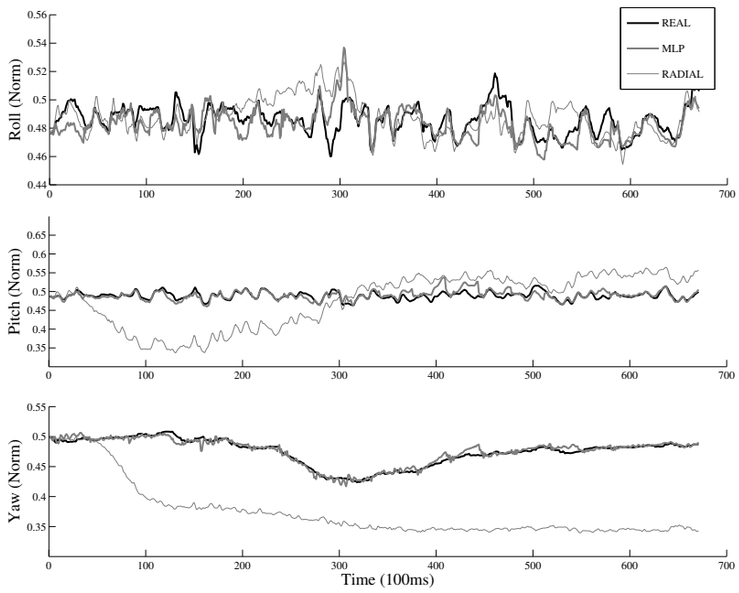


Figure 12.a. Real flight attitude vs. simulated flight attitude with MLP and RB

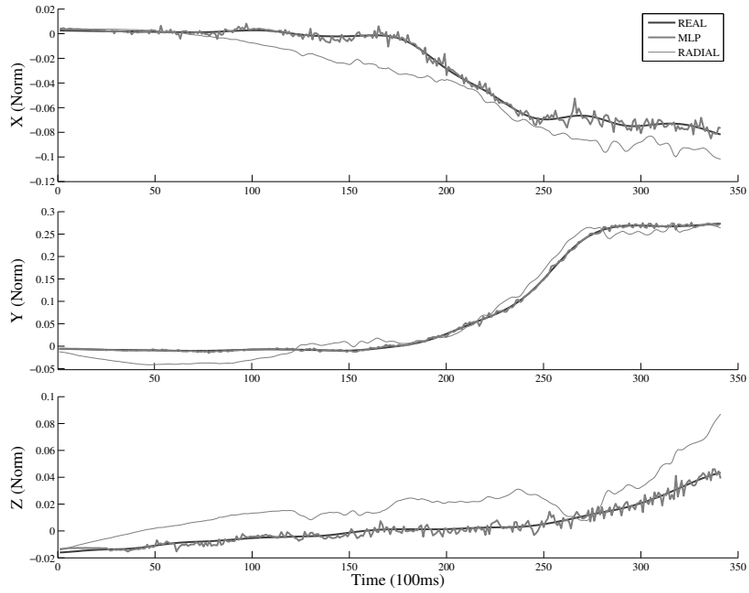


Figure 12.b. Real takeoff position vs. simulated takeoff position with MLP and RB

Table 3 summarises the errors for the degrees of freedom of attitude and position, respectively.

	Takeoff		Flight		Landing	
Attitude	MLP	Radial	MLP	Radial	MLP	Radial
Roll	4.17E-06	3.47E-07	8.53E-05	1.72E-04	3.13E-05	2.34E-05
Pitch	2.82E-06	3,.00E-07	9.57E-05	5.01E-03	4.76E-05	2.27E-05
Yaw	3.51E-06	4.23E-07	2.63E-05	1.18E-02	2.81E-05	6.14E-06
Position	MLP	Radial	MLP	Radial	MLP	Radial
X	8.13E-06	2.08E-04	2.06E-04	2.53E-01	8.66E-05	6.76E-05
Y	6.69E-06	3.39E-04	1.13E-04	6.37E-01	6.25E-05	9.25E-05
Z	7.33E-06	3.20E-04	1.09E-04	7.16E-03	8.43E-05	4.17E-05

Table 3.MSE for RB and MLP

In general the MLP shows a better performance, even though the RB signal tracking is remarkable. However the MLP requires 18 hours of training while the RB is trained in minutes. It is important to note that each simulation step is 'instantaneous' because it only requires few matrix multiplications.

It must be noted that RB networks have a better performance for local approximations (Chen et al., 1991) (Craddock & Warwick, 1996), which is confirmed by the data shown in Table 3 for the position simulation for take-off and landing.

6.3 Decoupled Training vs. Daisy Chain Training

Both scenarios require a trained attitude network which is the result of the previous sections analysis. The best attitude models are used with MLP or RB networks to simulate the position.

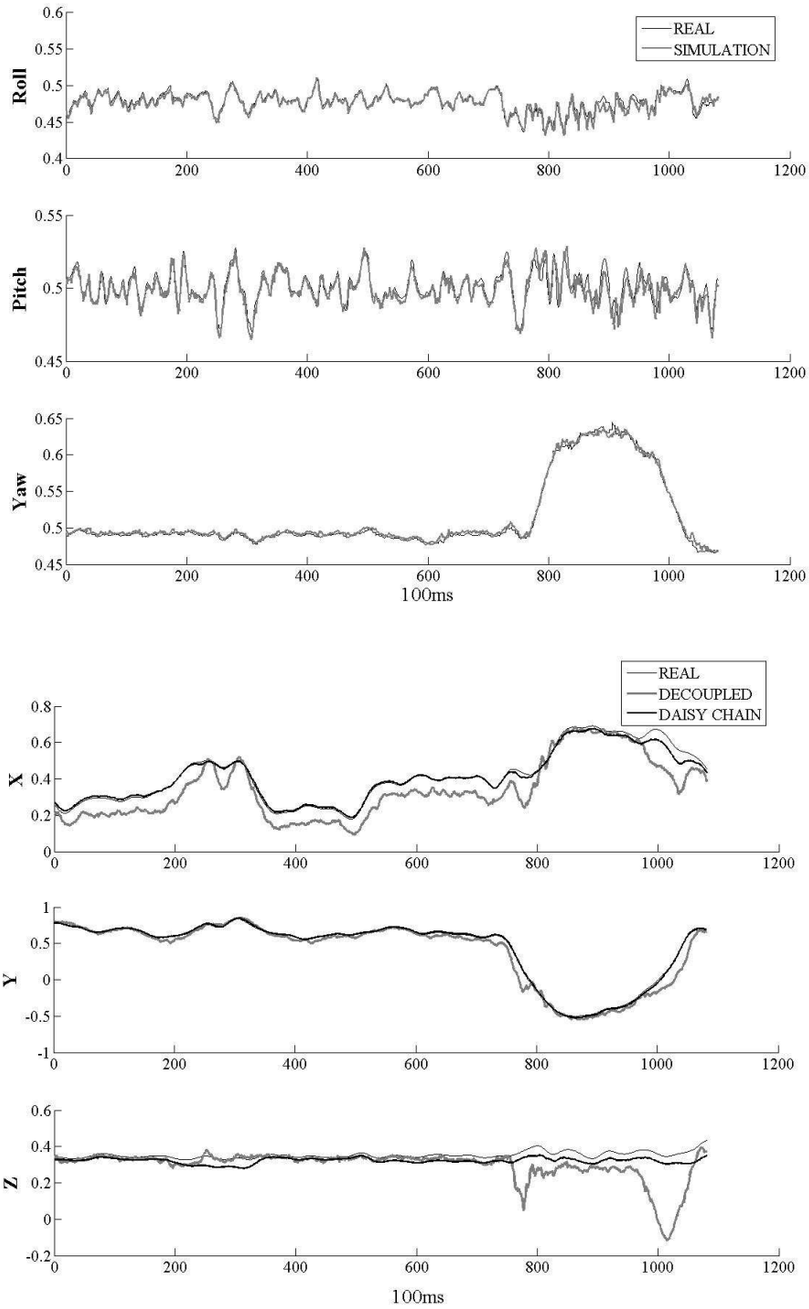


Figure 13. Attitude simulation for flight manoeuvres (Up) and position simulation for flight manoeuvres (down), with Daisy chain and Decoupled architectures

Fig. 13 shows the values of simulated versus real data for both systems (control-signals-to-attitude and attitude-to-position) for the two training architectures. Note that for the attitude simulation no distinction is made between the two architectures because the results are equivalent. Table 4 shows the quantitative results in terms of the MSE for six different flights sessions. It is important to note that these sessions are *not* the ones used for training. Both the graphic and quantitative results show that the daisy chain architecture has a better performance.

	Flight A	Flight B	Flight C	Flight D	Flight E	Flight F
Daisy chain						
X	1.50E-04	1.13E-04	2.13E-04	2.06E-04	1.25E-04	9.13E-05
Y	2.36E-04	1.44E-04	4.55E-04	1.13E-04	8.34E-05	1.49E-04
Z	8.58E-05	9.38E-05	8.23E-05	1.09E-04	1.09E-04	1.06E-04
Decoupled						
X	5.85E-03	2.00E-04	6.08E-03	5.99E-03	1.97E-04	3.87E-03
Y	1.28E-04	3.28E-03	6.35E-04	1.19E-03	9.20E-04	1.09E-03
Z	9.84E-02	3.77E-04	1.73E-02	6.85E-04	1.12E-04	2.55E-03

Table 4. Comparison of MSE for real vs. simulated position, for different flight sessions (only the flight manoeuvre stage is considered)

7. Conclusion

The main objective of this study was to show a new method for identifying dynamic and complex systems as in UAVs, by means of supervised neural networks. The methods described and the results obtained confirm the viability and performance of this alternative method.

It was observed that even though a decoupled architecture has a lower training error, the daisy chain architecture has better simulation performance. This is due to the fact that a network trained with a decoupled architecture does not consider the error propagated from the attitude simulation to the position model.

Empirical results show that the helicopter is modelled better if different flight stages are treated independently. One of the most significant variations in the vehicle dynamics is ground effect, a variation that cannot be accommodated by a single neural network. This means that different models are needed not only for take-off, landing and free flight, but also for particular manoeuvres in different physical and weather conditions.

In general, a better performance was observed with MLP, although the signal tracking achieved by the Radial Base network is remarkable. The main difference lies in the error rate, which in this case is better for MLP. But while an RB network needs 30 minutes training and yields the results mentioned above, MLP needs 18 hours.

It must be taken into account that Radial Basis networks have a better performance for local approximations, as shown in the fig. 6.2.b and table 3 corresponding to take-off and landing. On the other hand, global approximations are better with MLP. This is also shown in the

flight versus simulated data figures, where MLP exceeds the error rates of the Radial Basis networks.

The importance of this work lies in the confirmation of neural networks as a valid tool for dynamic system identification. Although this is only a first approach, the results obtained with this identification are very encouraging and an excellent basis for future work, where each flight stages have its hybrid networks like an fuzzy systems models each flight stages in Sugeno works (Nguyen & Prasad, 1999).

8. References

Chen, S.; Cowan, C.F.N. & Grant, P.M. Orthogonal Least Squares Learning Algorithm, for Radial Basis Function Networks: *IEEE Transactions on neural networks*, Vol.2, nº2, March 1991.

Craddock, R.J. & Warwick, K. Multi-Layer Radial Basis Function Networks An Extension to the Radial Basis Function: *IEEE International Conference on Neural Networks*, 1996.

Demuth, H. & Beale, M. Neural Networks TOOLBOX@ Version 4.0, The Math Works, USA, 2004.

Elman, J. Finding structure in time: *Cognitive Science,* Vol. 14, pp. 179-211, 1990.

Freeman, J.A & Skapura, D.M. Neural Networks. Algorithms Applications, and Programming Techniques, Addison-Wesley, Massachusetts, USA, 1991.

Hopfield, J. Neural networks and phisycal systems with emergent collective computational abilities: *Proceedings of the National Academy of Science, National Academy of Sciences*, Vol.81, pp. 3088-3092, 1982.

Jordan, M. Serial order: A parallel distributed processing approach: *Technical report, Institute for Cognitive Science*. University of California, USA, 1986.

Li, J.H.; Michel, A.N. & Porod, W. Qualitative Analysis and Synthesis of a Class of Neural Netwroks: *IEEE Trans. Circuits Syst.*, Vol.35. Nº 8, Aug.1988.

Lopez, J.L. Helicópteros: Teoría y diseño conceptual, ETSI Aeronáuticos, Madrid, Spain, 1993.

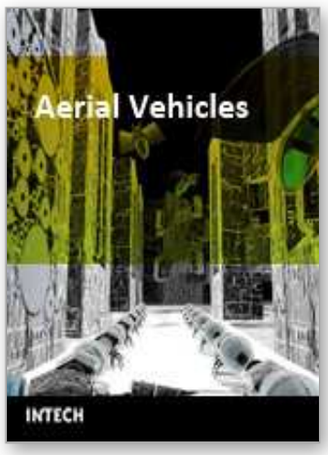
Narendra, K.S. & Parthasarathy, K. Identification and Control of Dynamical Systems Using Neural Networks: *IEEE Transactions on neural networks*, Vol.1 Nº 1, March 1990.

Nguyen, H.T. & Prasad, N.R. Fuzzy modelling and control, Selected works of M.Sugeno, CRC Press, 1999.

Norgaard, M.; Ravn, O.; Poulsen, N.K. & Hansen, L.K. Neural networks for Modelling and Control of Dynamic Systems, Springer-Verlag, London, UK, 2001.

Novatel, OEM4 Family of Receivers: User Manual- Volume 1 & 2. Installation and Operation, NovAtel Inc., Canada, 2002.

OSD (Office of the Secretary of Defense), *Unmanned aircraft systems (UAS) roadmap, 2005-2030*, Secretary of Defensa, USA, 2005.



Aerial Vehicles

Edited by Thanh Mung Lam

ISBN 978-953-7619-41-1
Hard cover, 320 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009

This book contains 35 chapters written by experts in developing techniques for making aerial vehicles more intelligent, more reliable, more flexible in use, and safer in operation.It will also serve as an inspiration for further improvement of the design and application of aeral vehicles. The advanced techniques and research described here may also be applicable to other high-tech areas such as robotics, avionics, vetronics, and space.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen