# Explorative data analysis of flight behaviour

Ivana Čapková, Michal Jůza, Karel Zimmermann

Czech Technical University
Faculty of Electrical Engineering
Department of Cybernetics

Summer stay at Delft University of Technology
Faculty of Information, Technology and Systems
Department of Mediamatica/Knowledge Based Systems

August 1, 2002

# Contents

# Acknowledgements

This work was carried out thanks to a grant of the FUNDATIE VAN DE VRIJVROUWE VAN RENSWOUDE TE DELFT for students exchange program between the Faculty of Electrical Engineering at Czech Technical University in Prague and Faculty of Information, Technology and Systems at Delft University of Technology. We would like to devote our acknowledgement to drs.dr. Leon J. M. Rothkrantz (Department of Mediamatica/Knowledge Based Systems at TU Delft) and Prof. Ing. Václav Hlaváč, CSc. (Department of cybernetics at CTU Prague) for realising of our stay at TU Delft. We would like also to thanks to drs.dr. Leon J. M. Rothkrantz and other people working for this department for valuable and helpful advices during our work and for careful check-ups of this report.

# Abstract

This paper describes 3 different approaches to the problem of situation estimation from outside-known parameters. We applied them on the problem of estimation of the pilot's goal using some parameters of the aircraft and controls used by the pilot. Our approach is to use recurrent (Elman) neural networks, load the above mentioned parameters in a time step into it and estimate which of the ahead-determined situation is applied at that time. The second approach is prediction of future values of these parameters from the parameters themselves. The third one is to smooth the values of the parameters in time and do further consideration from the trends (the consideration from these trends is not described here, however).

This paper devotes much space to data analysis done in various ways.

# Chapter 1

# Introduction

This paper describes a part of a project, which is aimed to support the pilot mainly in stressful or unusual situations. Advances in aircraft performance and weapons capabilities have led to a dramatical increase in the tempo of tactical situations facing the combat pilot, reducing the pilot's available processing and decision time. Technological advances in cockpit displays and electronics have resulted in an explosion in the complexity and sheer quantity of information that is available to the pilot. The pilot has more things to deal with in the cockpit (each of which is becoming more complex to understand), and less time to deal with them. This fact motivated the development of the advanced pilot/vehicle interface (PVI) concepts that will make optimal use of the pilot's capabilities, while recognizing his limitations. The PVI should enhance the information flow between the pilot and cockpit in such a way as to improve the pilot's *situation awareness* and decision-making, while alleviating the workload, and thus improve the pilot/vehicle systems survivability, lethality, and ultimately, its mission effectiveness. To meet these objectives, a clear understanding of the pilot's capabilities and limitations should drive the development of the interface concept. The objective of this project is to investigate the feasibility of developing interface concepts that adapt PVI content, format, and modality to meet the pilot's information needs as a function of the tactical situation and the pilot's workload level. The goal is to develop a PVI that knows what the pilot needs to know, when he needs to know it, and how to convey that information (i.e. via visual, auditory, or haptic interfaces).

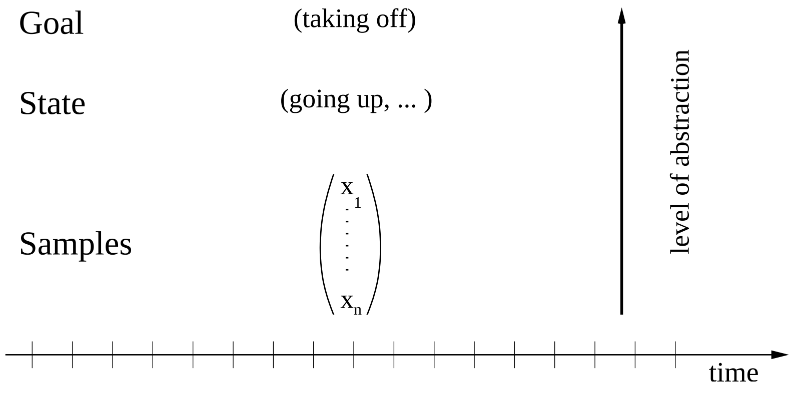


Figure 1.1: Illustration of levels of abstraction of the problem

At regular time we record the values of pilot-control (e.g. `throttle, brake, elevator`) parameters and feed back some parameters of the aircraft (e.g. `pitch-deg, roll-deg, acceleration`). The goal of this research is to give an interpretation of these data, what is the "planned" action of the pilot, what is his goal. As a proof of concept we limit ourselves to this set of actions: `going up, regular flight, turning right, turning left, going down, standing (on the ground), taxiing.`

In this report we present three different approaches:

- estimation of the pilot's goal from the control and aircraft parameters (with the use of recurrent neural networks),

- prediction of the values of these parameters from history (with the use of feed-forward neural networks),
- smoothing of the data so that trends can be seen and used for situation estimation (exponential smoothing).

For reaching the above described goals we used:

- **FlightGear** – generating data,
- **prepr,preprnew** – data preprocessing (written by ourselves),
- **SNNS** – handling neural networks,
- **Octave** – some of data analysis,
- **SPSS** – statistical data analysis.

# Chapter 2

# Data Acquisition – FlightGear Simulator

At this time there are many advanced flight simulator software tools available. But for the purpose described in the chapter 1, we need to manipulate input data and adapt the cockpit environment. For that reason we also need the source of the software (for the possibility of logging variables); but the commercial available tools don't provide the source so we decided to use free open-source FlightGear flight simulator ([1]).

The FlightGear flight simulator project is an open-source, multi-platform, cooperative flight simulator development project. The goal of the FlightGear project is to create a sophisticated flight simulator framework for use in research or academic environments, for the development and pursuit of other interesting flight simulation ideas, and as an end-user application. This framework can be expanded and improved upon by anyone interested in contributing. It is being developed through the gracious contributions of source code and spare time by many talented people from around the globe. The idea for FlightGear was born out of a dissatisfaction with current commercial PC flight simulators. A big problem with these simulators is their proprietariness and lack of extensibility. There are so many people across the world with great ideas for enhancing the currently available simulators who have the ability to write code, and who have a desire to learn and contribute. Many people involved in education and research could use a spiffy flight simulator frame work on which to build their own projects; however, commercial simulators do not lend themselves to modification and enhancement. The Flight Gear project is striving to fill these gaps. This paragraph is taken from [1].

This software makes it possible to log the following variables (various aircraft properties, controls used by the pilot, etc.):

```
ACCELERATIONS - nlf
CONSUMABLES - fuel
CONTROLS - aileron, aileron-trim, brakes, elevator, elevator-trim, flaps,
gear-down, mixture, parking-brake, rudder, rudder-trim, starter, throttle
ENGINES - cranking, fuel-flow-gph, oil-pressure-psi, oil-temperature-degf,
rpm, running, vac-psi
ENVIRONMENT - clouds, ground-elevation-m, visibility-m,
wind-from-down-fps, wind-from-east-fps, wind-from-north-fps
ORIENTATIONS - heading-deg, heading-magnetic-deg, pitch-deg, roll-deg
POSITIONS - altitude-agl-ft, altitude-ft, latitude-deg, longitude-deg
STEAM - adf-deg, airspeed-kt, altimeter-datum-mb, altitude-ft,
gyro-compass-deg, gyro-compass-error, horizont-adjust, mag-compass-deg,
slip-skid, turn-rate, vertical-speed-fpm
SURFACE-POSITION - elevator-pos-norm, flap-pos-norm,
left-aileron-pos-norm, right-ailerons-pos-norm, rudder-pos-norm
VELOCITIES - airspeed-kt, alpha-deg, mach, side-slip-deg, side-slip-rad,
```

`speed-down-fps, speed-east-fps`

and many others.

Figure 2.1: FlightGear – romantic flight above Los Angeles

Settings of the logging must be specified in the `preferences.xml` file in `<logging>` section. The *sample rate* is also possible to set in this section. We usually use a sampling interval of *1 sec.* When the session of the flight simulator is finished, logged variables are written into a text file (name of this file is also specified in `preferences.xml` file). Datasets we got in two ways (using *autopilot* feature or manually). We used autopilot because of its smoother courses of variables. Non-autopilot flights were done only by one person because that is not so easy to handle flying (the simulator is really realistic and it took us a long time to learn to fly). We took a lot of flights (about 40, with average lenght 15 minutes and couple of turning). A lot of flights finished with a crash. At the end we decided to use only nice flights (without crash and seeming to be similar to real pilot's flight). A general problem with datasets is that we don't know the right flying behaviour (as professional pilot do). Using this data it makes no sense to generate rules (induced by these data) and use an expert system approach. That is the reason we use neural networks to analyse this data. The results are of course dependent of training data, but the methodology not. So we can use neural networks as a proof of concept.

Figure 2.2: View from the cockpit

# Chapter 3

# Data Analysis

## 3.1 Introduction

In this chapter we'd like to bring up some data analysis of the flight data.

It can be expected that some variables are correlated. We can also expect that these correlations are action-dependent.

The data processed in this section are all acquired with the use of *autopilot* feature, sometimes they are also normalized or exponentially smoothed.

Some properties of the analyzed data got by PCA analysis:

- eigen values of the covariation matrix of the data:

$$\lambda = \begin{pmatrix} 0.02028 \\ 0.03427 \\ 0.08686 \\ 0.34380 \end{pmatrix}$$

  that data are mainly stretched in one direction, others are not very important,

- eigen vectors of the covariation matrix of the data:

$$v_1 = \begin{pmatrix} 0.038014 \\ 0.027285 \\ -0.108990 \\ 0.992941 \end{pmatrix}, v_2 = \begin{pmatrix} -0.085563 \\ 0.944142 \\ -0.313086 \\ 0.057034 \end{pmatrix}, v_3 = \begin{pmatrix} 0.216901 \\ -0.293391 \\ -0.925477 \\ -0.101827 \end{pmatrix}, v_4 = \begin{pmatrix} 0.971693 \\ 0.147560 \\ 0.183279 \\ -0.021138 \end{pmatrix}$$

so the coordinates of each data point in the PCA transformation $(p_1, p_2)$ are computed as

$$p_1 = x \cdot v_4 \tag{3.1}$$
$$p_2 = x \cdot v_3, \tag{3.2}$$

where $x$ is the set of `(normalized pitch-deg, throttle, normalized acceleration, normalized roll-deg)` ; $v_4$ and $v_3$ correspond to the largest eigen values.

This means that `pitch-deg` parameter has a large impact on $p_1$, which is little involved also by `throttle` and `acceleration` parameters, while `acceleration` parameter has a large impact on $p_2$, which is also involved by `throttle` and `pitch-deg` parameters and the impact of `roll-deg` parameter is rather small. That's the reason why `turning left` and `turning right` cannot be recognized from the PCA figure on fig. 3.2. As expected these variables are more or less independent. Next future we will use Sammon mapping and plot all the variables on 2 dimensions.

## 3.2 Basic view of the data

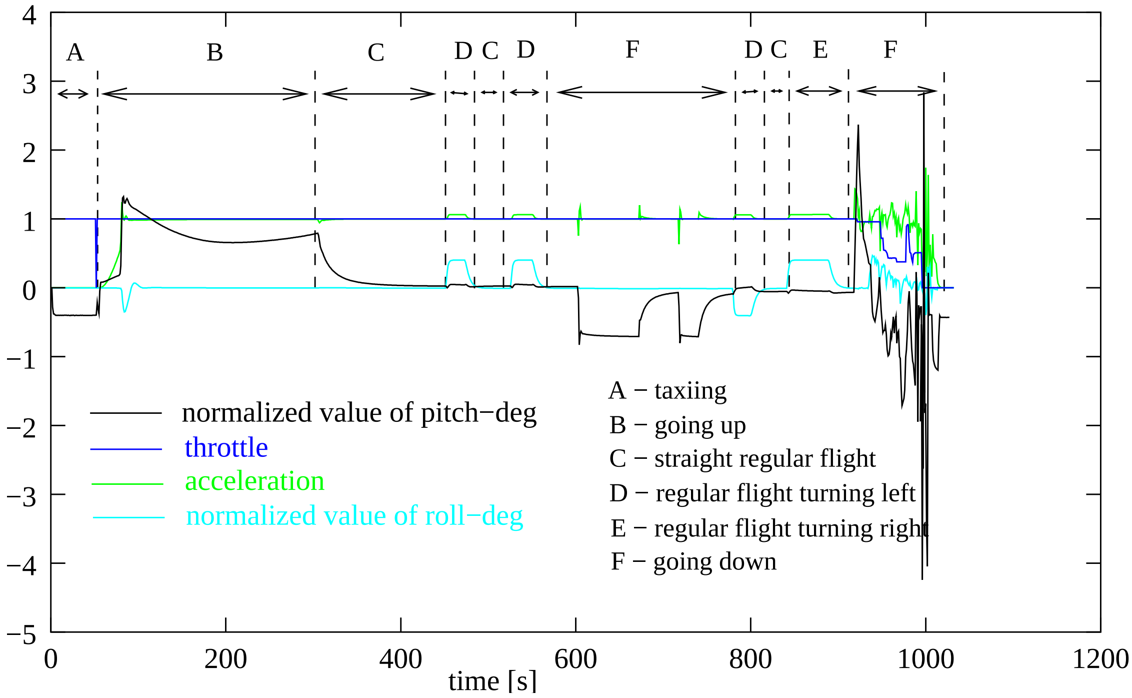


Figure 3.1: Time graph of selected flight variables during a simple flight (using autopilot feature).

We selected four variables to log (will be described later); the number at the beginning of the line stands for the normalization factor for each variable:

```
7 orientation/pitch-deg
1 throttle
1 accelerations/nlf
50 orientation/roll-deg
```

These normalization factors were determined just by looking at the data; we didn't make any precise analysis (reasons for these will be described later).

Figure 3.1 shows the time graph of the flight data.

Figure 3.2 shows the PCA and other projections of data acquired using the autopilot feature, so you can detect separate clusters.

Figure 3.3 shows the projection of variables' tracks during a simple flight. The left part of the figure stands for approximately first half of the flight and the right one for the rest of the flight. Before this analysis, data were exponentially smoothed, $\alpha = 0.03$.

The variable `roll-deg` lets us recognize the states of `turning left` and `turning right`, which can be seen very well on the projection of `pitch-deg` – `roll-deg` on fig. 3.2.

The variable `pitch-deg` lets us recognize the states of `going up` and `going down`, but this sometimes can be a problem of noise, as can also be seen on the figures. Variables `throttle` and `acceleration` are more complex, but finally they are able to recognize the rest of the states, as can be seen on fig. 4.2.

On the figures 3.2, 3.3 can be seen that there is a problem about data classification: the clusters lay over one another. This can be a problem of the data logging or maybe just the problem of a complicated interference of the levels of state and goal (see figure 1).

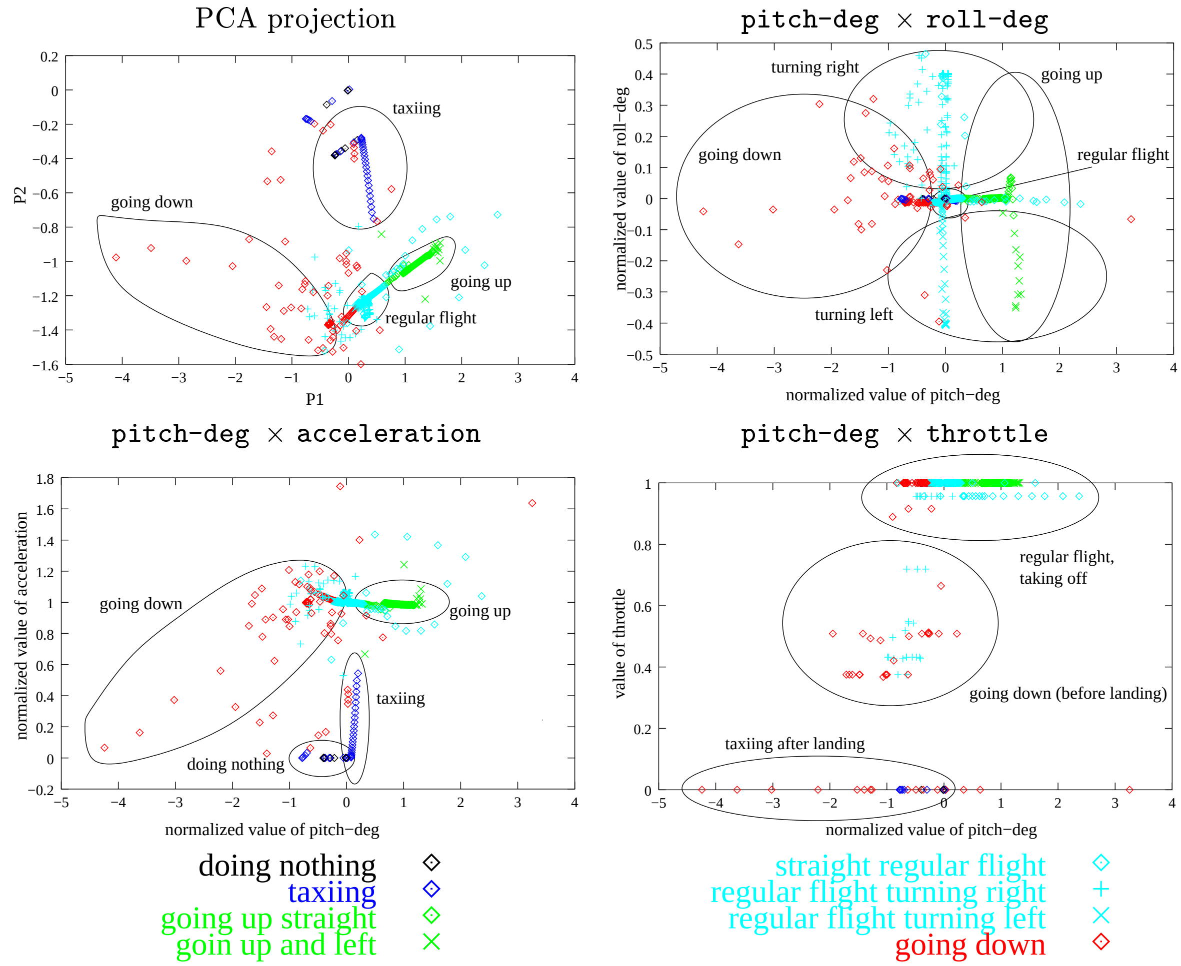


Figure 3.2: Various projections of flight data

## 3.3 Differenced cross-correlation functions

In this section we'd like to estimate some relations among various variables the FlightGear makes possible to log. We consider the differenced cross-correlation to be more important than regular cross-correlation because we are mainly interested in trends. Also the course of regular cross-correlation is much worse (we didn't put a figure here) – doesn't have peaks at all, going down slowly with the lag.

These relations helped us to figure out what variables to log and what not to. Figures 3.4, 3.5, 3.6 show the cross-correlation functions of the variables. These cross-correlation functions are made of non-normalized data (whole flight). Cross-correlation can be figured out by

$$cor(x, y, d) = \frac{\sum_i (x(i) - E(x))(y(i-d) - E(y))}{\sqrt{var(x)var(y)}} \tag{3.3}$$

where $x$ and $y$ are (differenced in our case) time courses of selected variables and $d$ is the lag which changes.

At fig. 3.4, 3.5 you can see that `acceleration` with `elevator`, `acceleration` with `pitch-deg` etc. are strongly correlated, compared to fig 3.6 which shows the weakly correlated variables of `throttle` with `turn-rate` and `pitch-deg` with `turn-rate`.

From cross-correlation function courses (fig. 3.4, 3.5) dependence among most important variables can be seen (e.g. `acceleration` – `elevator` cross-correlation function has maximum in the lag equal to zero - that means that `acceleration` and `elevator` parameters are correlated and the dependency isn't delayed. On the other hand, `roll-deg` – `turn-rate` cross-correlation function has in the lag equal to minus one – they are dependent and the `turn-rate` parameter is one second "delayed" in

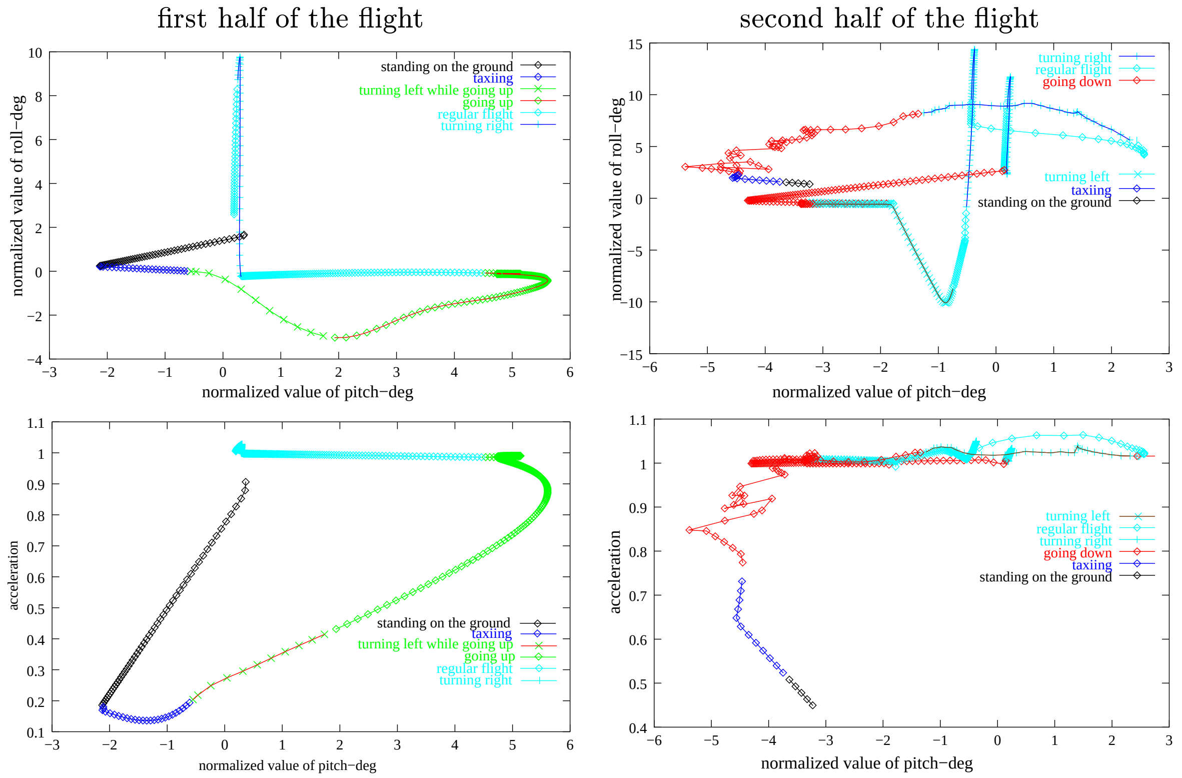


Figure 3.3: Courses of parameters of the flight – smoothed data

comparison with `roll-deg` parameter).

From cross-correlation functions courses (fig. 3.6) it is clear that there is no important dependency between the `throttle` and `turnrate` or `pitch-deg` and `turn-rate` parameters. Unimportant dependecy mean that maximums are under the confidence limits.

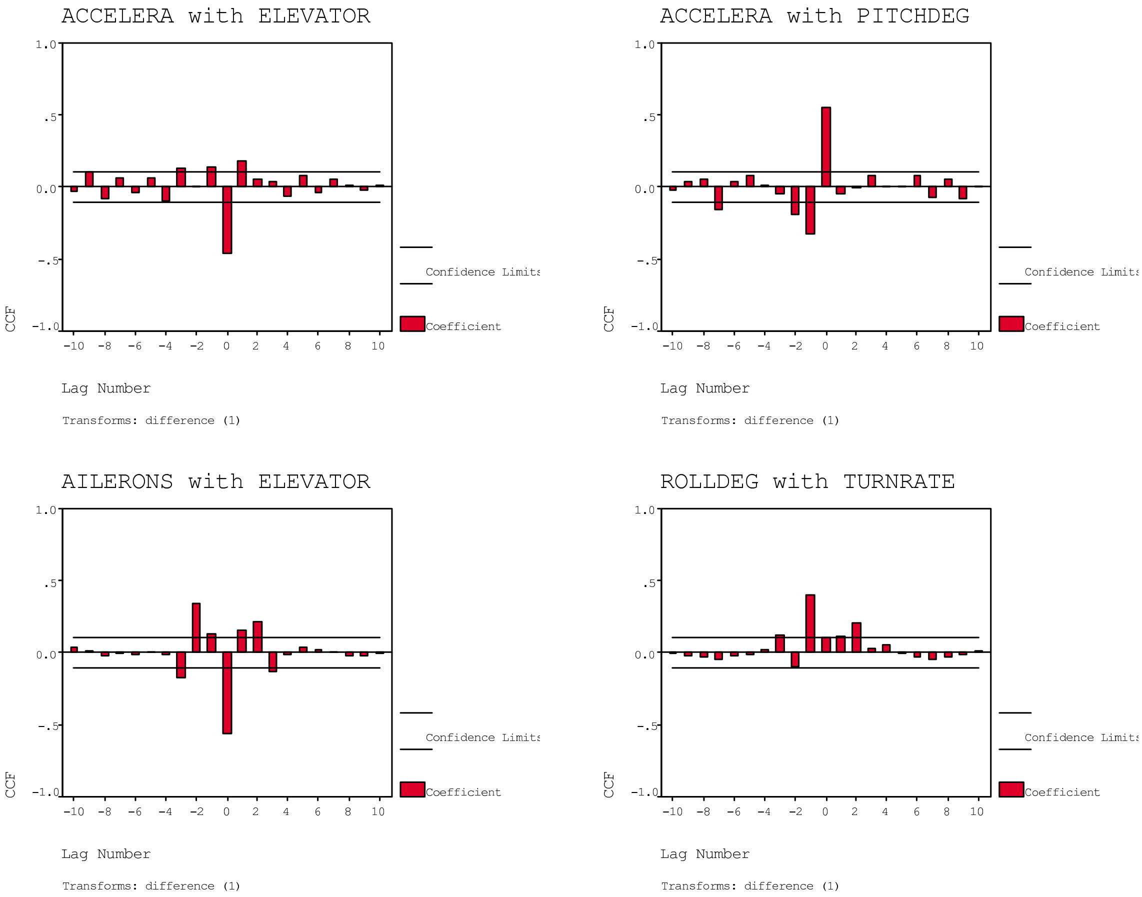


Figure 3.4: Differenced cross-correlation functions of some variables logged by FlightGear simulator – related variables

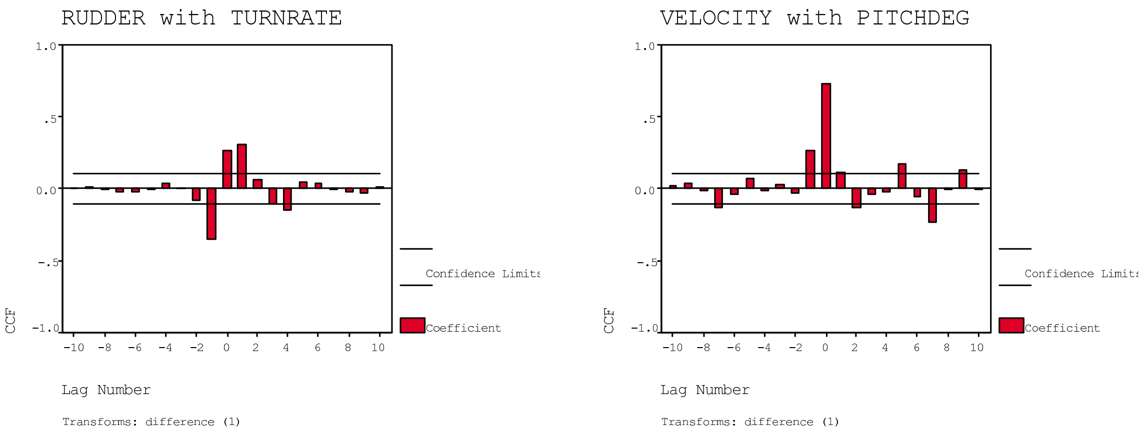


Figure 3.5: Differenced cross-correlation functions of some variables logged by FlightGear simulator – related variables

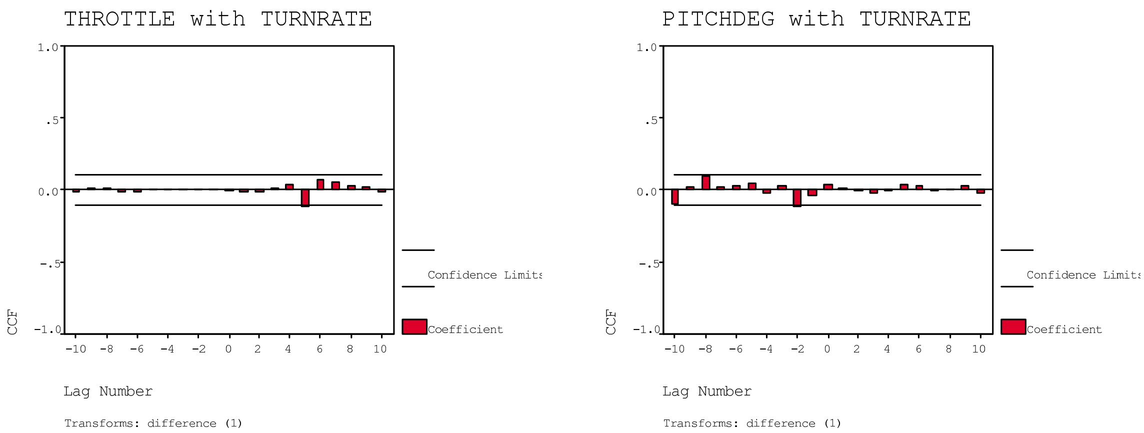


Figure 3.6: Differenced cross-corelations functions of some variables logged by FlightGear simulator – not much related, as can be seen from the figure

# Chapter 4

# Assessment of the Activity/Action and Goal of the Pilot

## 4.1 Approach

Our purpose is to explore if it is possible to use neural networks (NNs) for assessment of action/situation. We decided to discriminate only among seven basic situations (described below). We use (partly) recurrent NN (*Elman*) network because we suppose a time context exists among situations. We generate the data using *FlightGear* flight simulator (See [1]). For handling NNs we use *Stutgart Neural Network Simulator (SNNS), version 4.2* (See [2]).

For training our neural network we of course need classified data. To be able to attribute the desired network output to the input data, we also manually logged pilot's actions during time. Problem of FlightGear logging was that although we set the *sample rate* to *1 sec* , the variables were logged in the period of about *1.2 sec* . Handling of this problem will be described later.

## 4.2 Data Preprocessing

Because the structures of the `.log` file from the FlightGear simulator and `.pat` file to be loaded into SNNS as a pattern file are different, we made a program for data conversion and preprocessing. We also needed to put the desired outputs of the network to the `.pat` file. So for every flight we created another file, named `l.act`, where `l` stands for the name of the logfile without the extension (this means if our logfile is named `data.log`, we should create a file `data.act`) and put in the codes and durations of various epochs of our flight. Our program than writes these patterns to the `.pat` file. The structure of the `.act` file will be described below. But training of neural network can sometimes be a problem if the input variables are from different ranges, so our program also provides a simple normalization. That means you can divide your variable with a float number you can choose (but you cannot substract the mean value as you may like). If normalization should be made your current directory must include the file `l.nrm` , where `l` stands for the name of the logfile without extension again. Structure of this file will be described below.

### 4.2.1 Structure of the `.log` File

Structure of the `.log` file is probably not so interesting because you get this file from your FlightGear simulator. This logfile should contain names of all variables logged on its first line and the values of all these variables in columns (in the same order). The first column corresponds to *time* and these values are not written to the `.pat` file. For some variables, FlightGear logs just the *false* or *true* values – our program puts corresponding values *0* and *1* to the `.pat` file.

Sometimes it proves to be very useful to check the `.log` file because the first two lines are often bad (some values are missing), which causes an error message in the SNNS program.

### 4.2.2 Structure of `.act` File

The sign '%' stands for a comment continuing till the end of the line.

The first data in the file is *total time* of the flight in seconds, minutes etc. – what's most convenient for you. All other time data must be in the same units. This approach also solves the problem of logging the sample rate as described above by computing the ratio of the time of this action to the total time.

The second data in the file is the number of outputs, which is recommended to be the same as the number of recognized actions.

Then follow the desired outputs of the neural network for each action and behind the multicolumn `";"` there is the duration of the action. It is important that the sum of all the particular durations equals the total time written in the first row of the file.

There's no rule about repetition of each action but if these data don't correspond to the `.log` file, the network would be trained badly.

The example of an `.act` file is the following:

```
%total time
21
%number of outputs
7

%action; time of action
0 0 0 0 0 0 1;1
1 0 0 0 0 0 0;5
0 1 0 0 0 0 0;10
0 1 1 0 0 0 0;5
```

### 4.2.3 Structure of `.nrm` File

The first line stands for the number of variables which should be normalized. Sure, all variables may be said to be normalized and for those ones which shouldn't be really normalized there can be `1` in the `.nrm` file, but it's much faster if it's not done this way.

On the other lines put the normalization factor first and then the name of the variable. This name must correspond to that in your `.log` file.

Your data will be *divided* by this normalization factor.

An example of an `.nrm` file is the following:

```
7
40 velocities/vertical-speed-fps
3 accelerations/nlf
30 orientation/pitch-deg
50 orientation/roll-deg
100 steam/air-speed-kt
2 steam/turn-rate
0.2 controls/elevator-trim
```

### 4.2.4 Function of `prepr` Program

The purpose of this program is to automatically create the `.pat` file suitable to be loaded to the SNNS program. When putting

```
./prepr data
```

on the command line, the program will take the files `data.log, data.act` and `data.nrm` (if it's available) and create a file `data.pat` to be loaded as a pattern file to SNNS.

## 4.3 Situations to Recognize

We decided to recognize only the following situations (example of the coding is in the brackets):

```
going up  (1 0 0 0 0 0 0)
regular flight (0 1 0 0 0 0 0)
turning right (0 0 1 0 0 0 0)
turning left (0 0 0 1 0 0 0)
going down (0 0 0 0 1 0 0)
standing (on the ground) (0 0 0 0 0 1 0)
taxiing (0 0 0 0 0 0 1)
```

There are also some possible combinations of these basic actions, e.g. when you fly normally (that means you're neither going up nor down) and turn left at the same time or turning when still taking off (going up). We chose this kind of coding because we thought that it's good for the neural network to have as many different states as possible and recognizing the values of 0 and 1 is much simpler than recognizing continuous values.

## 4.4 Elman Network

Elman network is a partially recurrent neural network. The connections are mainly feed-forward but also include a set of carefully chosen feedback connections that let the network remember cues from the recent past. The input layer is divided into two parts: the true input units and the context units that hold a copy of the activations of the hidden units from the previous time step. As the feedback connections are fixed, backpropagation can be used for training of the feed-forward connections. This paragraph is taken from [4]. An example of Elman network with one hidden layer is on fig. 4.1.

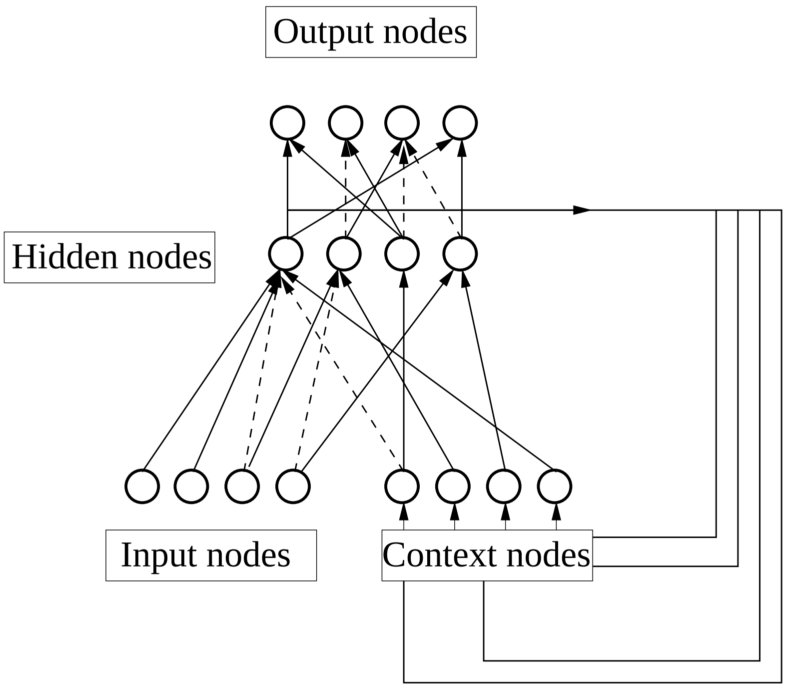


Figure 4.1: Simple Elman network – 4 input units, 1 hidden layer with 4 units, 4 output units, 4 context units

## 4.5 Project Progress

**Experiment 1:**

First thing we tried was to create some data and load them into a neural network. We logged 17 variables, didn't normalize them at all, had 5 situations to recognize (first 5 in the list) so we also needed just 5 output variables. We tried to train both Elman and Jordan networks, the latter seeming more logical, the former having slightly better results. After that we decided to use just Elman network, with output context.

Discouraged by very bad results, we tried to separate the state of `standing on the ground` so we added one more output variable. Our Elman neural network had 2 hidden layers with 30 units each and training took a very long time. We initialized the network with 0 at the self-recurrent links and 1 on the recurrent links from hidden and output layers. Teaching was being made with the teaching outputs on the input (not the real ones) so the training error was much lower than the testing one.

**Experiment 2:**

After we decided to normalize our data. One approach certainly is to compute a mean and variance of each data set and normalize them so they have zero mean and standard deviation equal to 1. We decided not to use this approach because we supposed that a recognizer can be run real-time in future. It's also unusable to normalize each data set to different mean and different variance, which would have far different effect.

With these data, we tried to train networks with various numbers of layers, various numbers of units in each layer, with and without output context (that means whether the output layer also has its own context units) – it seems that output context is important and so is the number of layers. For example, a network with just one hidden layer with five units and with output context had much better performance than the network with one hidden layer with five units and without output context.

**Experiment 3:**

After the data analysis with the help of PCA (Principal Component Analysis) we decided to log just 3 variables – `pitch-deg, roll-deg` and `throttle`. Once we logged `vertical-speed` instead of `pitch-deg`, but `pitch-deg` is much more smoothed. The problem of PCA is that it emphasizes the variables having largest variance, not considering the importance of the changes (which the system of course couldn't know), e.g. when a variable has just a small variance but the changes are very important, this variable is not considered very important. **Experiment 4:**

We still had problems with `throttle` variable – every flight used different levels of throttle, so we added a more objective variable – `acceleration`. For better precision in recognition of `standing` and `taxiing` we left `throttle` variable in our data sets.

**Experiment 5:**

Having problems about the quality of training data (which is much more important than the quality of testing data), we decided to use the *autopilot* feature in the FlightGear simulator. The neural network trained with these data proves very good results, even for the data from non-autopilot flight.

**Experiment 6:**

We consider this approach to be final so the particular network and results are described in the next sections.

## 4.6 Our neural network

We decided to use Elman neural network with 2 hidden layers, 10 units each and output context. As should be assumed, our network has 4 units in the input layer, 7 units in the output layer and each context layer has the same number of units as the corresponding hidden or output layers.

For initialization, we use the built-in `JE_Weights` function with parameters $\alpha = -1$, $\beta = 1$, where $< \alpha, \beta >$ is the range at which the feedforward lines will be (randomly) initialized, $\lambda = 0$, which are the weights of the self-recurrent links of the context neurons, $\gamma = 1$, which are the weights of the links from hidden (and output) layers to the context units – these weights don't change during learning, and $\psi = 0$ which is the initialization value of context units. For more details, see [2]. For learning, we use just regular back-propagation, with the step width $\eta = 0.2$, error on which it shall stop $d_{max} = 0.1$ and $x = 0.8$ which is the coefficient of mixing real and teaching outputs during training (this means that the real output has larger impact). For updating, we use just regular `JE_Order` function.

## 4.7 Results

Figure 4.2 shows the training and testing error of the network, where both training and testing data were acquired using autopilot. Data sets for training and testing were different.

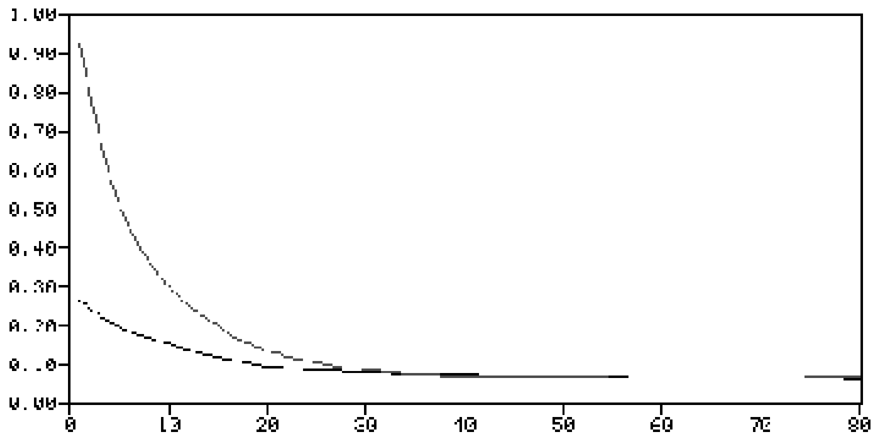


Figure 4.2: Training (lower) and testing (upper) mean square error (on different data)

Figure 4.7 shows the training and testing error of the network, trained on the data acquired from autopilot and tested on data from a simple manual flight. The testing error is higher in the beginning but lower at the end. For illustration, we denounce the percentage of badly classified patterns (we consider the significantly highest value of all the outputs):

**serious mistakes**

1.4 %, e.g. turning left when the pilot didn't turn left at all etc.

**less important mistakes** – delayed performance, mean delay 17.0 patterns, standard deviation 7.6 patterns.

We attribute these mistakes mostly to unprecised logging and some of it also to the recurrency of the network.

Testing error on data acquired with the use of *autopilot* feature:

**serious mistakes** 13.5 %

**less important mistakes** – mean delay 11.6 patterns, standard deviation 5.8 patterns.

## 4.8 Conclusions

It seems that it is possible to recognize the state of the aircraft (e.g. going up, going down) from the variables we decided to log, with the use of NN. The problem of recognizing the goal of the pilot is more philosophical – it's certainly correlated with the state of the aircraft but not totally. It much depends on the individuality of the person.

Nevertheless, when recognizing the state of the aircraft we needed to put lots of apriory information to our classifier but we still think it was neccessary. Training of neural networks was very complicated and took a long time, so we suppose that logging many variables was very unsuccessful. Also the normalization factors are given a priori – but they are here just for shifting the input data into a reasonable range.

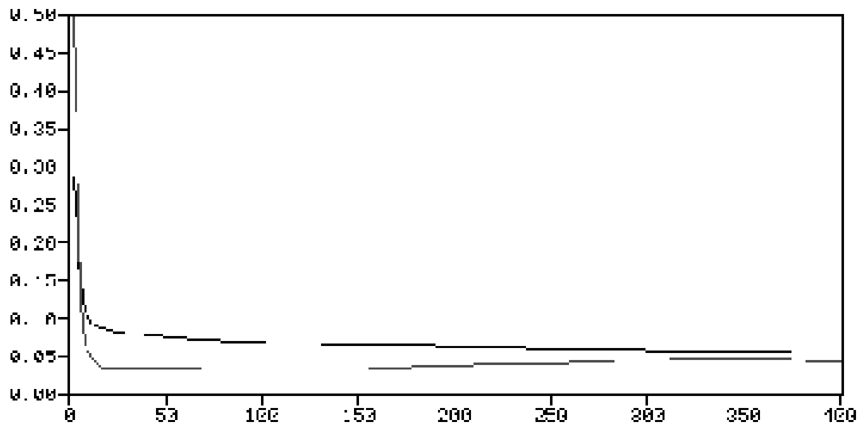

Figure 4.3: Training (higher at the end) and testing mean square error (simple manual flight)

# Chapter 5

# Predictions of Flight Variables Using Neural Networks

## 5.1 Introduction

In this chapter we'd like to predict the future values of the logged parameters. To make this, we need to know the previous values of these parameters. We decided to use NNs once more, just feedforward ones this time. There are many variations of this approach:

- the size of **history time window**, which means how many past values will be considered, can be chosen,
- the size of **future time window**, which means how many future values of a single variable should be predicted, can be chosen,
- what **input flight parameters** are to be used, which means from which flight parameters the predictions should be made,
- what **output flight parameters** are to be used, which means of which flight parameters the predictions should be made.

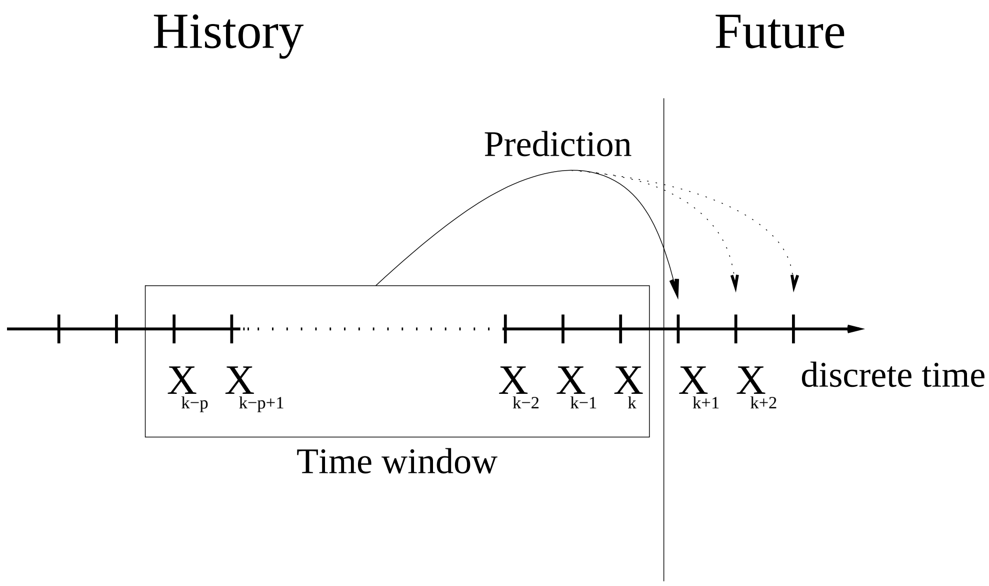


Figure 5.1: Prediction

We'll use a regular feedforward network with standard back-propagation learning method. Parameters of learning will be mentioned later.

In this section, for training the network we use the same data as shown in fig. 3.1, for testing the network other data, generated with the use of *autopilot* feature.

## 5.2 Data Preprocessing

For data preprocessing, we made a program similar to that one used for the approach described in the last section. This new program only needs `.nrm` file and of course also the `.log` file, but needs more parameters on the command line. The first parameter, like in the other case, means the name of all the files without extension, the second parameter is a binary determination of the input variables (from which the prediction is being made), the third parameter is a binary determination of the output variables (prediction of which variables is to be made), the fourth parameter is the input window size (i.e. how many input values of one variable are to be applied to the neural network) and the fifth parameter is the output window size (i.e. how many values of the output variables are to be predicted). The example of running our program is as follows:

```
./preprnew data 01100 01000 10 5
```

This means that the program will take the file `data.log` and `data.nrm` (voluntary) and put the results into the `data.pat`. The input variables of the network will be 10 past values of the second and third variable logged in the `.log` file (please remember that `time` is always the first variable logged in the `.log` file) and teaching outputs will be 5 future values of the second variables.

## 5.3 Learning conditions of our neural networks

We trained the network on a training set which is about 15 minutes long (detail description of flight is on fig.3.1). Sampling rate was 1 sec, that means slightly more that 1000 patterns. We planned several experiments with various size of time window(5,20...). We used regular feed-forward network, standard back-propagation learning method used in all cases, step size 0.2 in most cases. Learning 1000 epochs in most cases, no overtraining appeared.

Weights initialized randomly in all cases, in the range of $\langle -1, +1 \rangle$.

Both training and testing data normalized, with the same factors used, in all cases.

Output layer and the last hidden layer have pure linear activation function to provide the continuous outputs, others have sigmoidal function in all cases.

## 5.4 Results

### 5.4.1 One variable one-step prediction from one variable known, small time window

**Network description:** 2 hidden layers, 5 units each, time window size 5.

**Pitch-deg:**
MSE 0.01719 train error, 0.01026 test error See more details on fig. 5.2.

**Acceleration:**
MSE 0.02847 train error, 0.00564 test error See more details on fig. 5.3.

**Roll-deg:**
MSE 0.00115 train error, 0.00029 test error See more details on fig. 5.4.

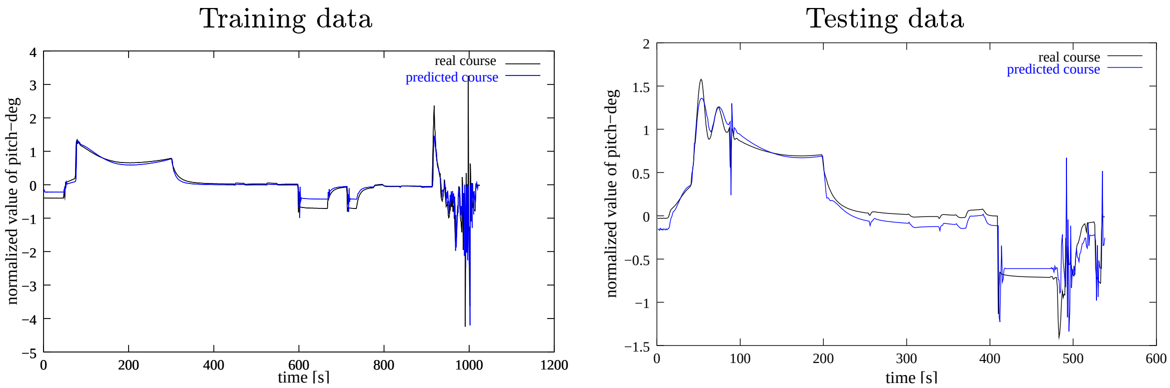


Figure 5.2: Prediction of pitch-deg variable, time window size 5

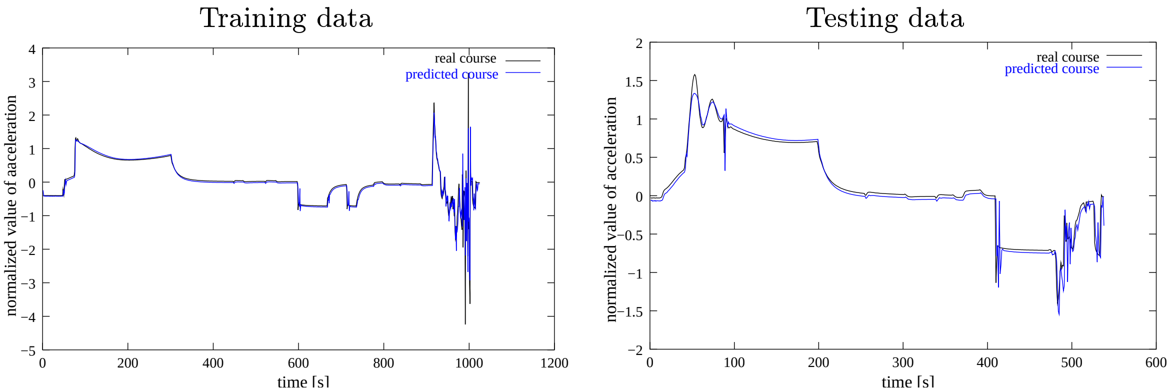


Figure 5.3: Prediction of acceleration variable, time window size 5

### 5.4.2 One variable one-step prediction from one variable, large time window

**Network description:** 2 hidden layers, 20 units each, time window size 20.

**Pitch-deg:**
MSE 0.16532 train error, 0.12102 test error See more details on fig. 5.5.

**Acceleration:**
MSE 0.00454 train error, 0.00995 test error See more details on fig. 5.6.

**Roll-deg:**
MSE 0.00538 train error, 0.00791 test error See more details on fig. 5.7.

### 5.4.3 One variable one-step prediction from all variables

**Network description:** 2 hidden layers, 10 units first one, 5 units the other one, time window size 5.
All 4 variables used for prediction, i.e. 20 input units.

**Pitch-deg:**
MSE 0.00329 train error, 0.11834 test error See more details on fig. 5.8.
(Step size learning parameter decreased to 0.1 for technical problems.)

**Acceleration:**
MSE 0.00170 train error, 0.02623 test error See more details on fig. 5.9.

**Roll-deg:**
MSE 0.00047 train error, 0.00760 test error See more details on fig. 5.10.

### 5.4.4 All variables one-step prediction from all variables

**Network description:** 2 hidden layers, 10 units each, time window size 5.
All 4 variables used for prediction, i.e. 20 input units, 4 output units.

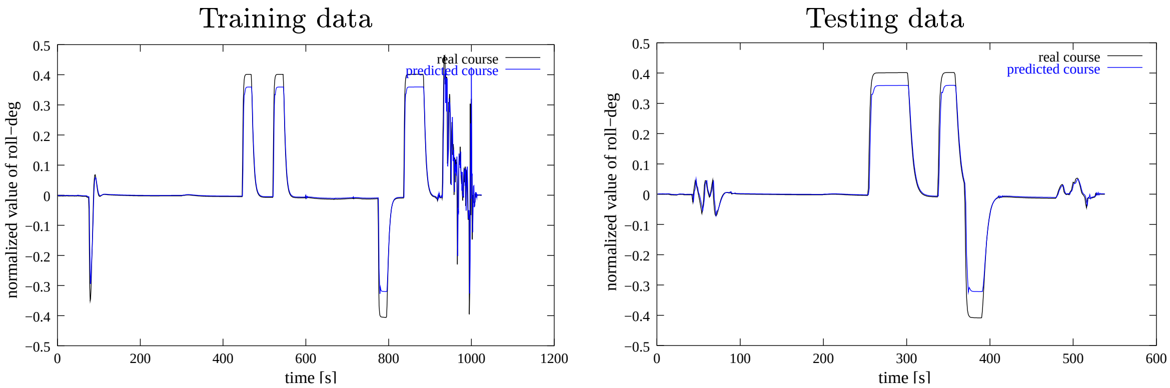


Figure 5.4: Prediction of roll-deg variable, time window size 5

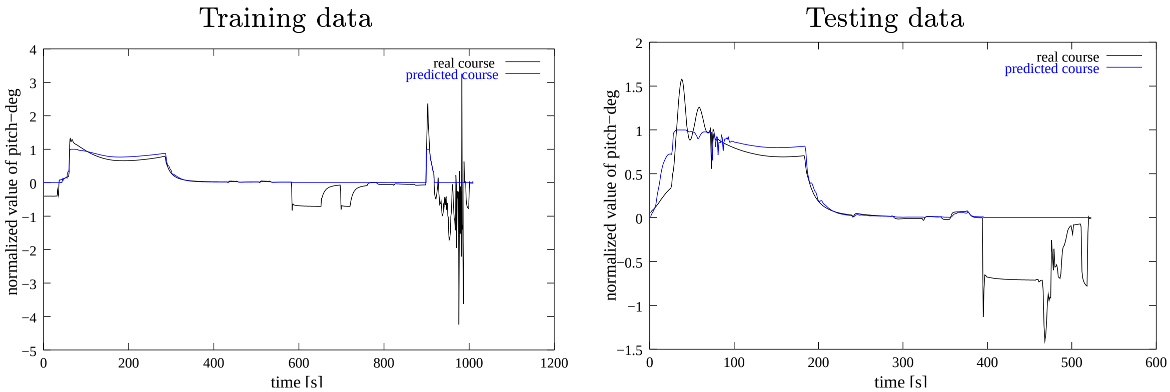


Figure 5.5: Prediction of pitch-deg variable, time window size 20

Step size 0.1, 1500 epochs (technical problems, divergence of the network)
MSE: 0.01334 train error, 0.09575 test error See more at figure 5.11

## 5.5 More steps ahead prediction of the `pitch-deg` variable

**Network description:**

2 hidden layers, 10 units first one, 8 units the other one, 10 units in the input layer, 5 units in the output layer, i.e. there are 10 past values of the variable on the input and 5 future values of the variables are the teaching output of the network.

MSE: 0.19639 train error, 0.62649 test error

learning parametres: 7000 epochs, 0.05 step size

Particular prediction performance can be seen at figures 5.12, 5.13, 5.14.

## 5.6 Conclusions

The one variable one-step prediction with the history time window size of 5 is practically working, except for bad levels of the signal, though the timing is right (maybe a little delayed, but it can't be seen at the figures). We consider these results to be quite good.

Large history time window has a worse performance, which can be attributed to the bigger complexity of the neural network and more difficult learning. We think that the real "order" of this system is not much higher than 1 and the black-box approach (NNs) has problem recognizing the "importance" of each of the values of data.

One-step prediction of one variable from all variables is disappointing too, because we supposed that more information would do the predictions better. The problem of even worse performance than

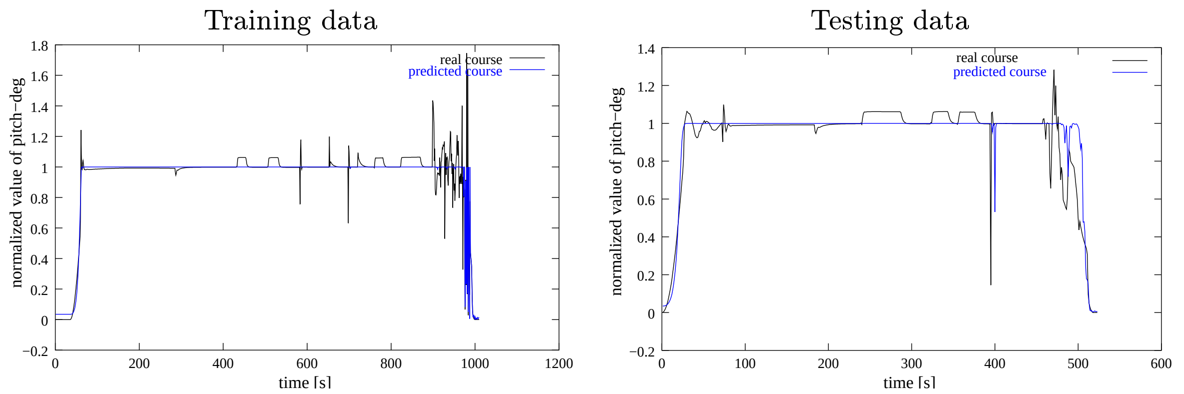


Figure 5.6: Prediction of acceleration variable, time window size 20

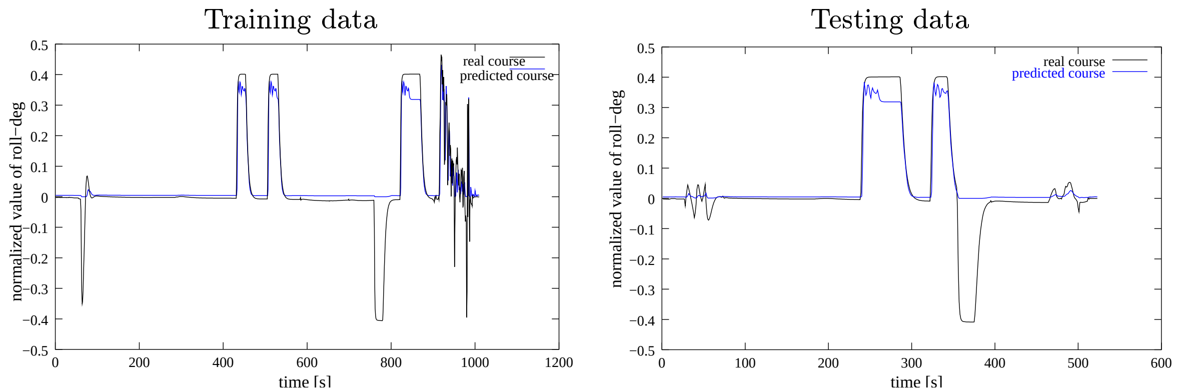


Figure 5.7: Prediction of roll-deg variable, time window size 20

the one of prediction from just one variable can also be attributed to the complexity of the network, maybe longer learning would be needed.

The same applies to prediction of all variables from all variables.

We didn't try to predict one variable from others at all because we think our variables are not highly correlated.

More-step prediction performance corresponds to the expectation, that means that the prediction error is getting bigger with the number of steps predicted ahead. Another problem in this case is the great difference between performance of training and testing data, but this can probably be reduced using more training data.

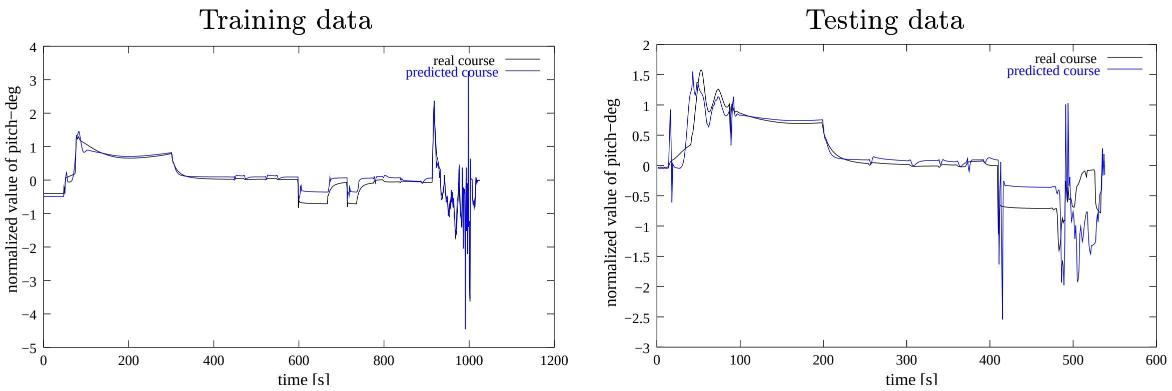


Figure 5.8: Prediction of pitch-deg variable from all variables, time window size 5

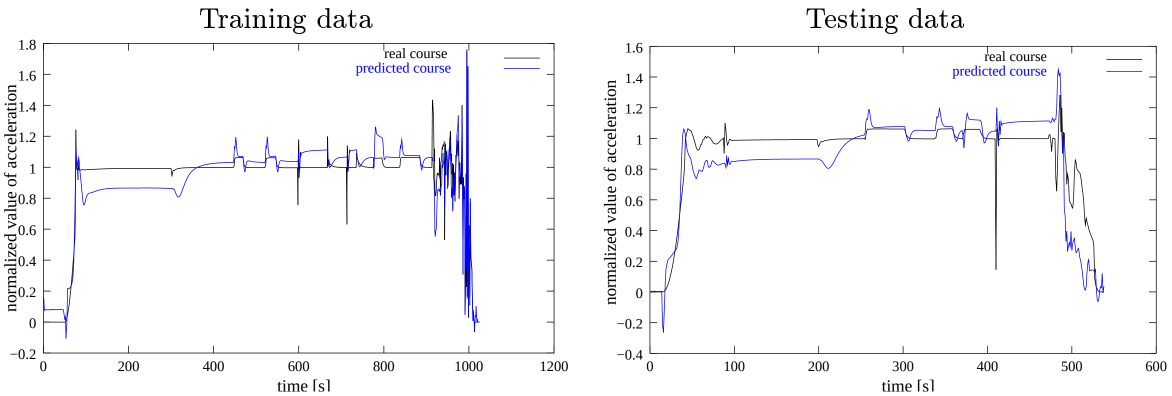


Figure 5.9: Prediction of acceleration variable from all variables, time window size 5

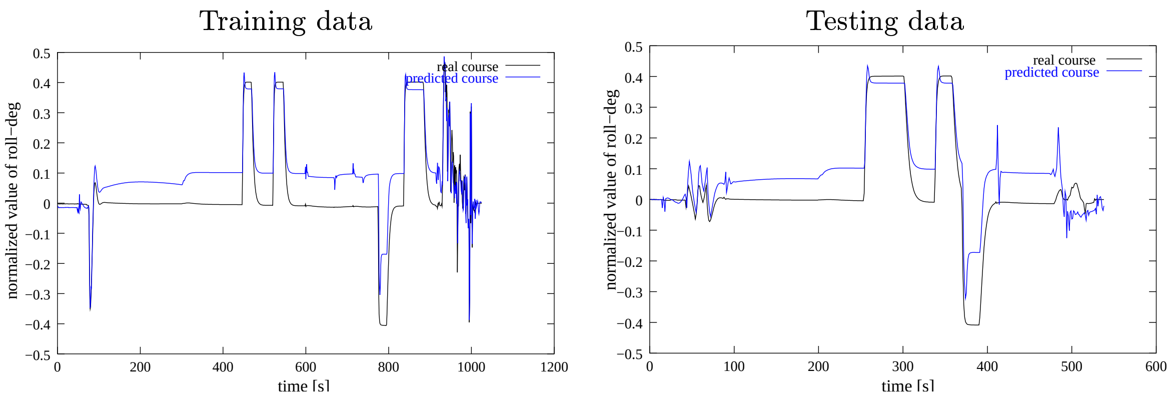


Figure 5.10: Prediction of roll-deg variable from all variables, time window size 5

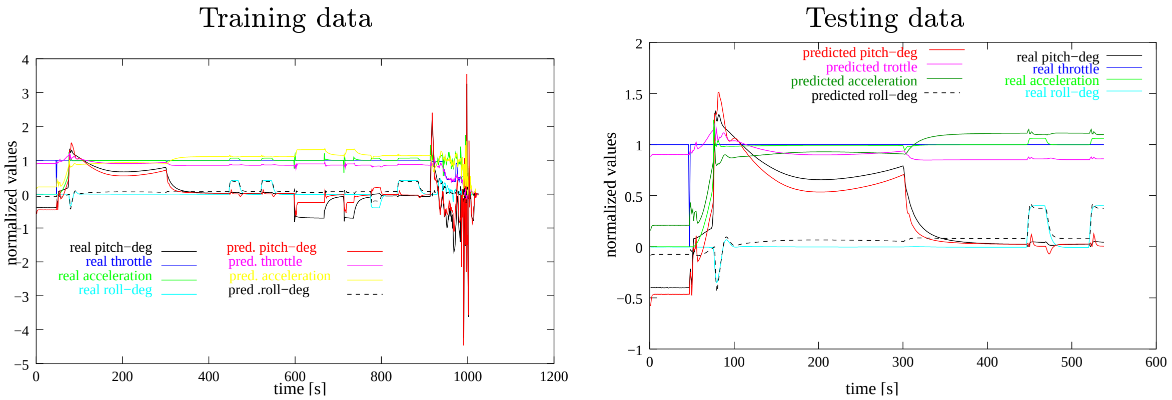


Figure 5.11: Prediction of all variables from all variables, time window size 5

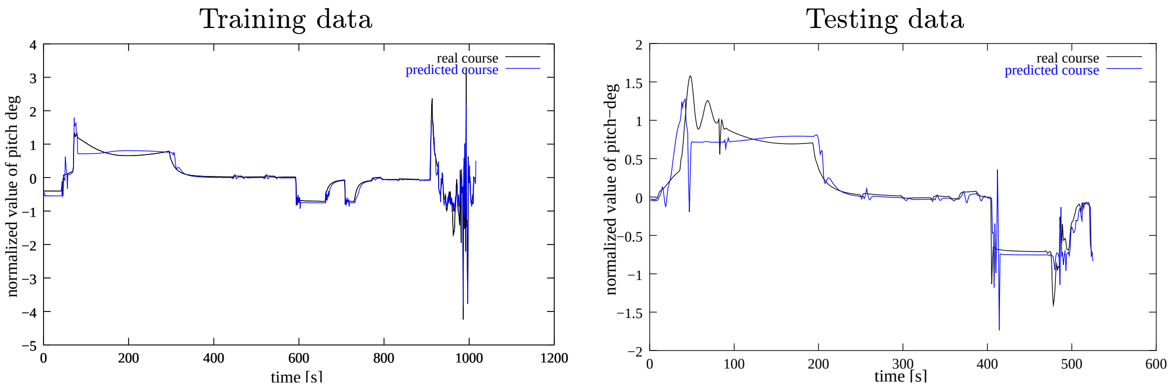


Figure 5.12: 1-step ahead prediction performance (`pitch-deg` )

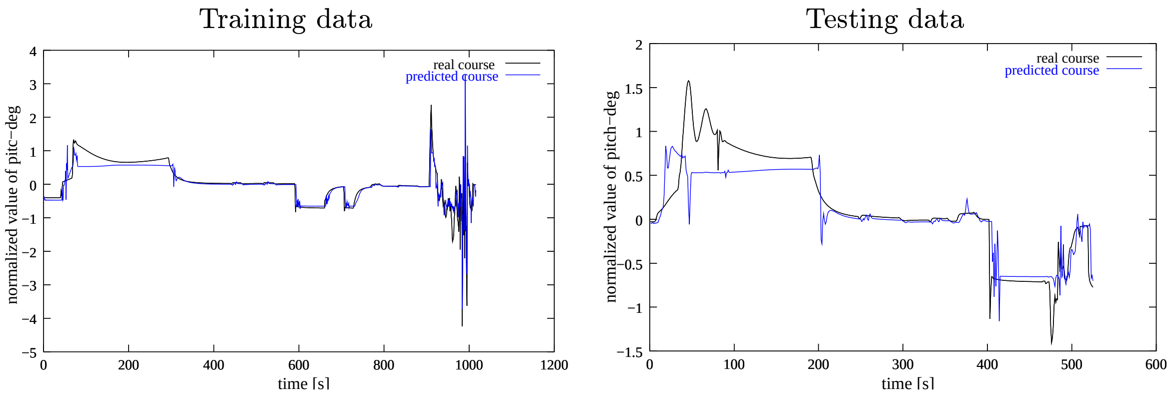


Figure 5.13: 3-steps ahead prediction performance (`pitch-deg` )

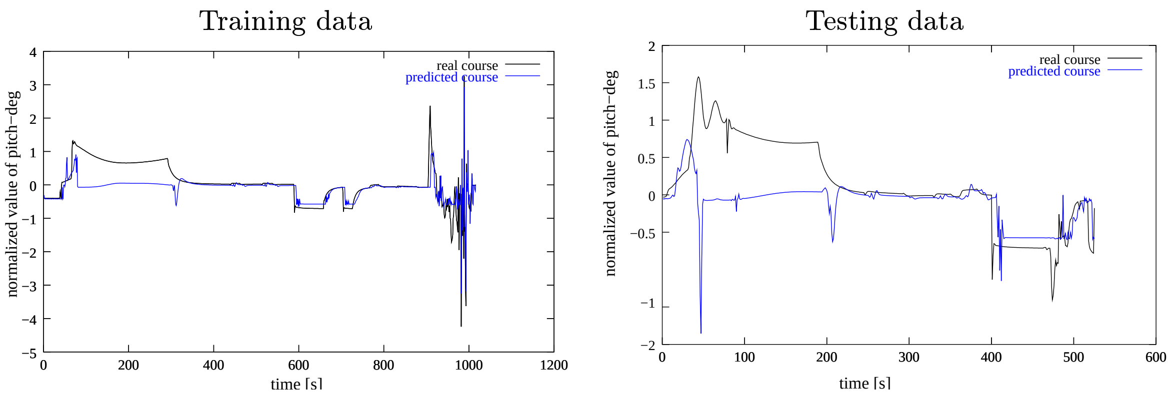


Figure 5.14: 5-steps ahead prediction performance (`pitch-deg` )

# Chapter 6

# Exponential smoothing

From the logging of the variables we can see a lot of peaks and valeys in the graphs, caused by stochastic behaviour. We want to smooth the graphs to detect the trends. We used SPSS-software for our time serie analysis. Results of EX-smooth you can see on fig.6.1 and on fig.6.2. There is the obvious conflict in courses of velocity and acceleration (velocity is falling down but the sign of acceleration is positive). The reason is that the FlightGear allows to log only the absolute value of acceleration .

$\alpha = 0.05$; MSE 10.03 $\alpha = 0.10$; MSE 6.45

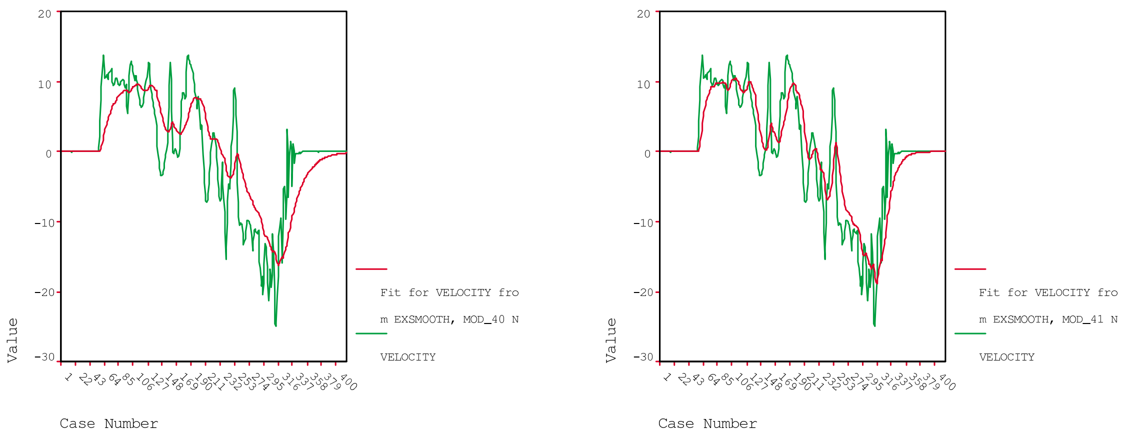


Figure 6.1: Exponential smoothing of the `velocity` variable using different values of $\alpha$

## 6.1 Conclusions

It is clear that smoothed data are better for the training of the NN and classifying an action/state in expert systems etc. Of course, the applicability of the exponencial smoothing depends on a good choice of $\alpha$ factor so there were always chosen two example values of $\alpha$ for each variable (signal). Results received by SPSS smooothing procedure (fig. 6.1,....,6.8) show that exponencial smoothing could be used to detect trends in the signals with large stochastic noise.

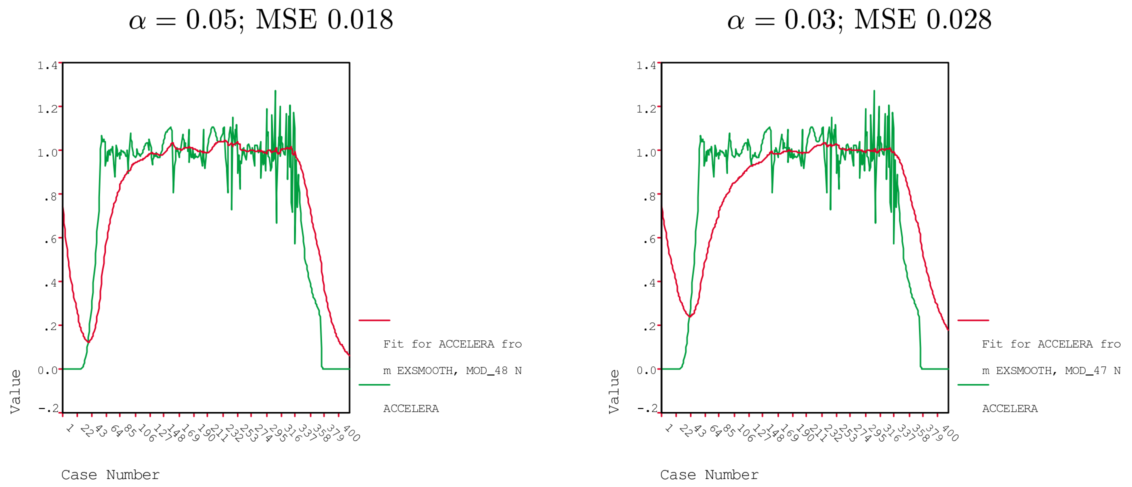


Figure 6.2: Exponential smoothing of the `acceleration` variable using different values of $\alpha$,()

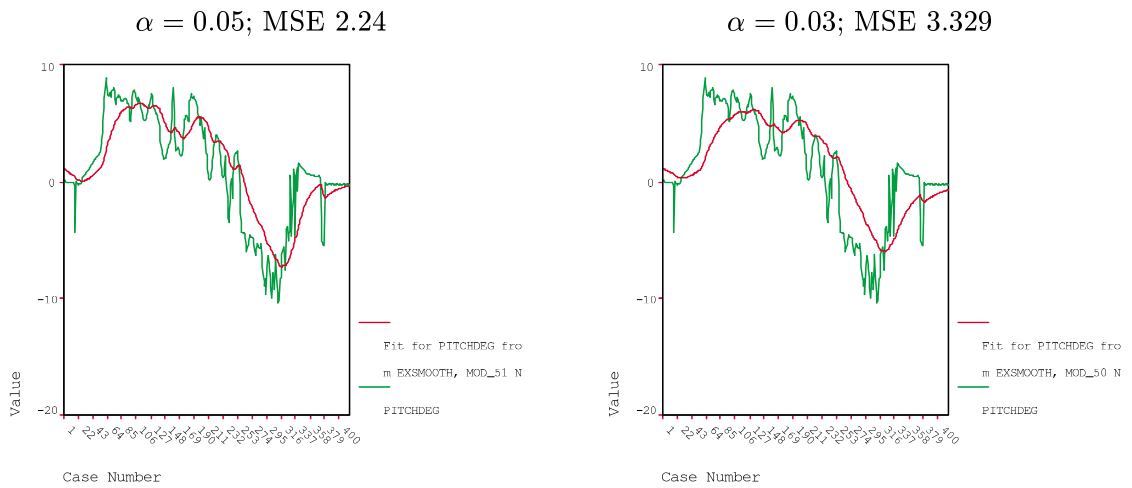


Figure 6.3: Exponential smoothing of the `pitch-deg` variable using different values of $\alpha$

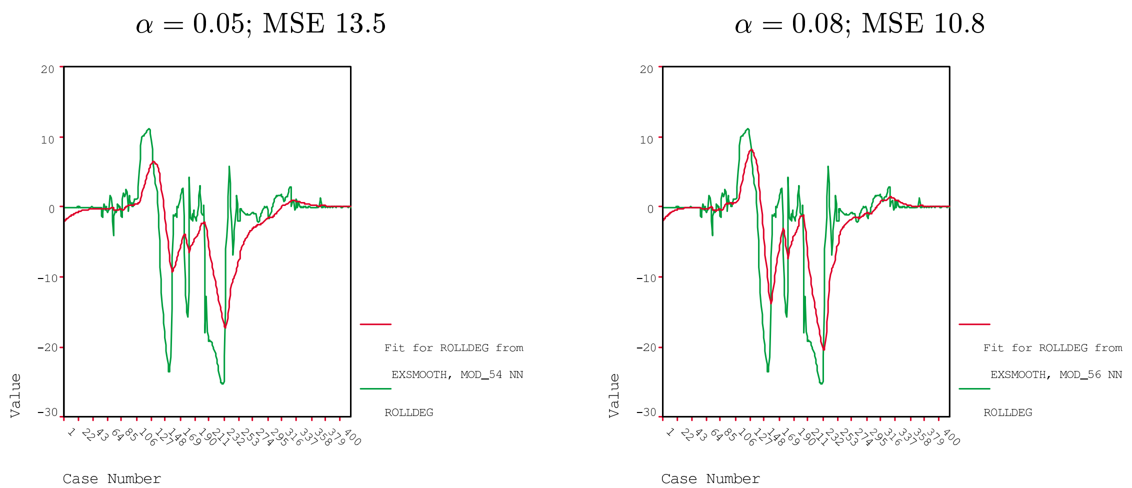


Figure 6.4: Exponential smoothing of the `roll-deg` variable using different values of $\alpha$

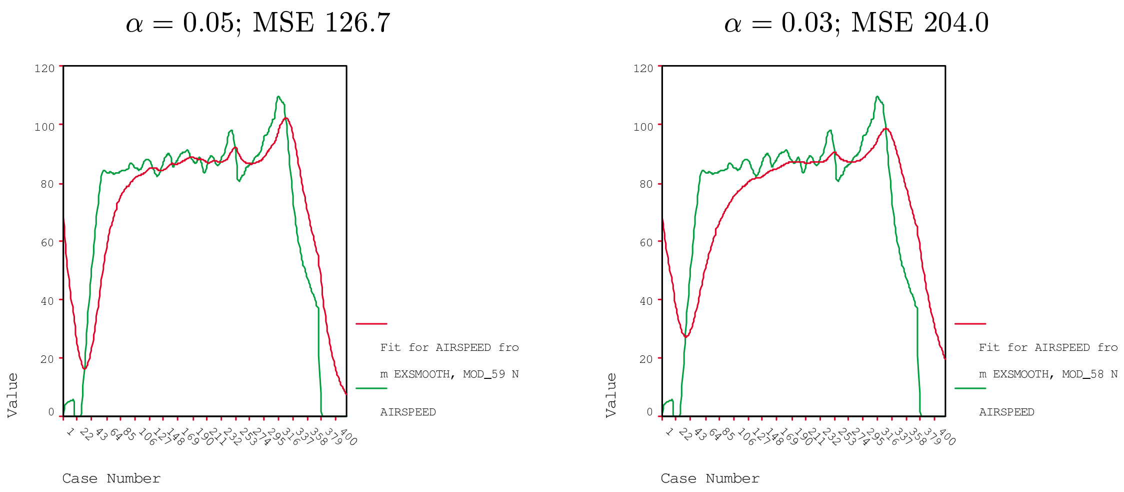


Figure 6.5: Exponential smoothing of the `air-speed` variable using different values of $\alpha$

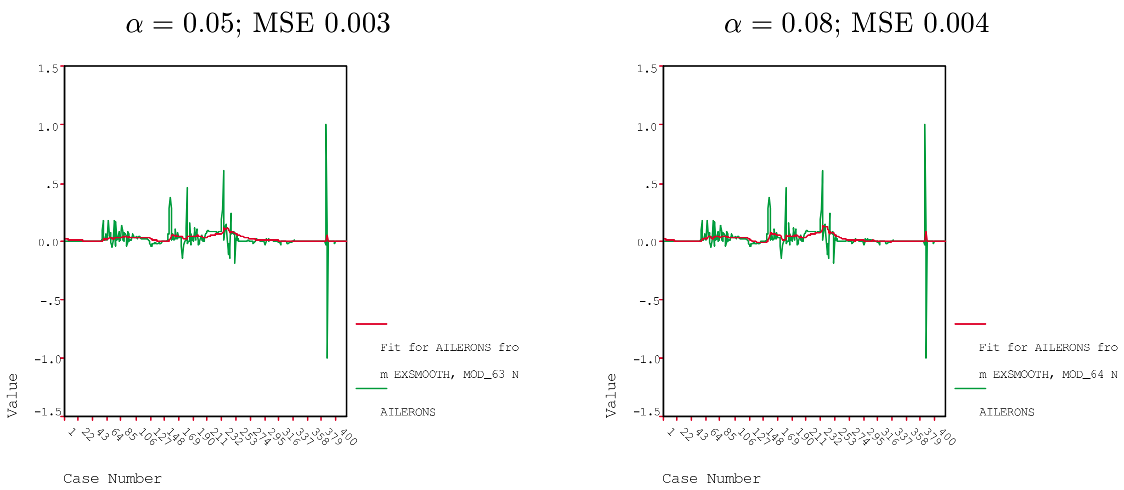


Figure 6.6: Exponential smoothing of the `ailerons` variable using different values of $\alpha$

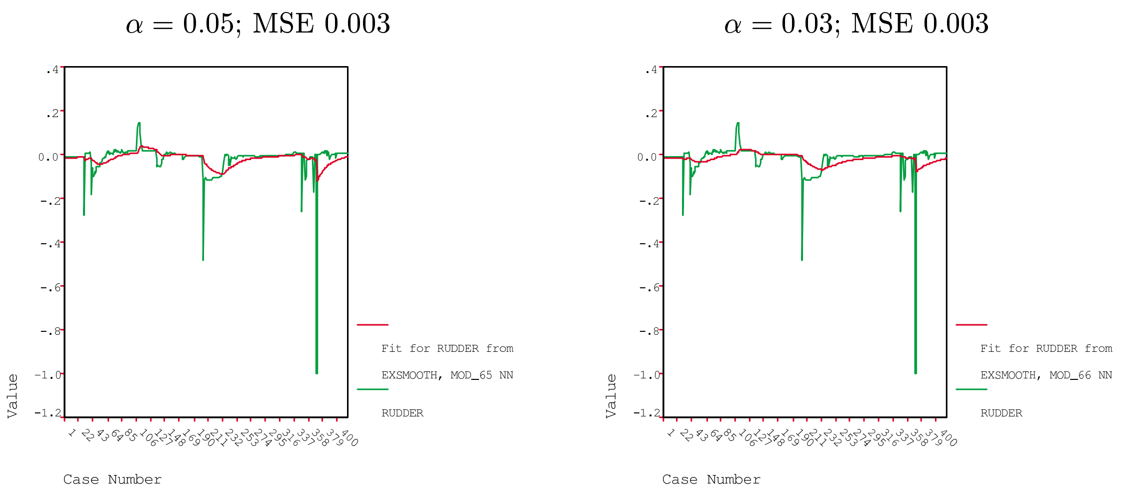


Figure 6.7: Exponential smoothing of the `rudder` variable using different values of $\alpha$

$\alpha = 0.05$; MSE 0.001 $\alpha = 0.08$; MSE 0.006

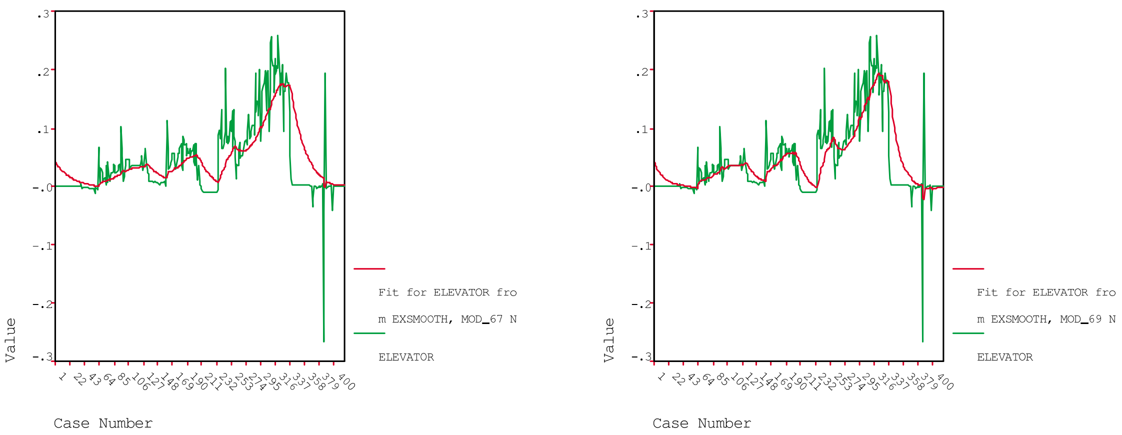


Figure 6.8: Exponential smoothing of the `elevator` variable using different values of $\alpha$

# Chapter 7

# General Conclusions

The goal of this work was to explore the possibilities of using the data from the flight simulator and support the decision about project's (PVI concept) future (if it is suitable to use NN for prediction and action/state estimation or if the expert system would be better etc.).

- Pilot's action estimation – we have to mention that we didn't differ between *goal* and *state* (see fig. refintro). In spite of this fact we don't recommend to use neural network for pilot's action estimation. It seems to us that expert system would be better for this purpose.
- Prediction – We think that it is possible to use forward NNs to predict one step ahead values of some variables although the prediction is not excellent. Another disadvantage is that you have to carefully set the size of the time-window otherwise the results are unusable.
- Trends – Exponential smoothing is a suitable tool for detecting trends. But again there is a problem with setting of $\alpha$ parameter which strongly influences the results.

# Chapter 8

# Future work

This article stands just for a beginning of a project of adaptive cockpit. It's purpose is just to sketch which approaches are possible to use and how successful they are. That means there's lot of work to be done in the future: e.g. combination of the parameters prediction and situation assessment from these parameters, loading the exponentially smoothed data into an expert system or just predict future values of the parameters from the ex-smoothing coefficients and much more.

# Bibliography


[1] FlightGear Simulator, http://www.flightgear.org

[2] Stuttgart Neural Network Simulator, http://www-ra.informatik.uni-tuebingen.de/SNNS/

[3] Mulgund, S. S., Zacharias, G. L.: A Situation-Driven Adaptive Pilot/Vehicle Interface, www.cra.com/publications/papers/hics96.pdf

[4] Elman Recurrent Neural Network, http://divcom.otago.ac.nz/ infosci/ kel/ software/ ricbis/ elman/ elman_main.html

[5] Intro to Time Series, http:// oll. temple. edu / economics / notes / timeseries / Timeseri.htm

[6] Autocorrelation, http:// www.itl.nist.gov / div898 / handboook / eda / section3 / eda35c.htm

[7] Autocorrelation Analysis, http:// www.csu.edu.au / ci / vol02 / cmxhk / node10.html

[8] SPSS/PC+ Trends manual, ISBN 0-918469-44-9