ACE project

Adaptive Cockpit Environment

Research Report

Explorative data analysis of flight behaviour with neural networks

Ivana Čapková
Michal Jůza
Karel Zimmerman
Patrick Ehlert

Knowledge Based Systems group
Delft University of Technology
The Netherlands

WP 6 & 10 Report

Reference: ACE/NLR/WP6 & 10

Internal reference: DKS02-04 / ACE 02

Date: September 2002

The ACE Consortium

NLR (Amsterdam, The Netherlands), Military Academy (Brno, Czech Republic), Delft University of Technology (Delft, The Netherlands)

For the Dutch Ministry of Defence

DOCUMENT IDENTIFICATION

Results of the Literature Study		**ACE/NLR/WP6 & 10**	
written by:		reviewed by:	
Name	Organisation	Name	Organisation
I. Čapková, M. Jůza, K. Zimmermann and P.A.M. Ehlert	Delft University of Technology	P.A.M. Ehlert and L.J.M. Rothkrantz	Delft University of Technology

DOCUMENT CHANGES

Issue date	**Version**	**Comments**
September 2002	1.0	

STATUS OF THE DOCUMENT

Internal	
Restricted	
Public	**X**

WORKGROUP
KNOWLEDGE BASED SYSTEMS
DELFT UNIVERSITY OF TECHNOLOGY

I. Čapková
Ir. P.A.M. Ehlert
M. Jůza
Q.M. Mouthaan
Drs. Dr. L.J.M. Rothkrantz
Ir. B. Sletterink
K. Zimmermann

Contents

Preface

This report is part of a project called Adaptive Cockpit Environment (ACE). In this report we describe our attempts to use neural networks for situation recognition and prediction.
The first chapter gives a short introduction to the ACE project and the problem that it tries to solve. The second chapter describes the data that was collected in order to perform our experiments. Chapter 3 deals with how we analyzed the data and were able to detect different states. In chapter 4 we show the training of a neural network to assess the current situation. Chapter 5 deals with the neural networks again, but now as a method to predict future situations. Finally in chapter 6 we give some general conclusions and talk about future work.

Abstract

This report describes our investigation of flight simulator data and our approach of situation recognition and prediction using artificial neural networks. We use aircraft and pilot control parameters and apply them to the problem of estimating the current situation. The parameters are fed to a recurrent Elman neural network that returns the current state. A regular feedforward network is used to predict future values of several parameters.

Chapter 1

Introduction

1.1 Problem statement

Due to the recent technological advances in aircraft performance and weapons capabilities, the time available to a pilot in a military aircraft has reduced significantly. Also, the amount of information available to a pilot and the complexity of the contents have increased. The pilot needs to assess a situation and make decisions in a split of a second. For this situation awareness (SA) is important, which means that the pilot needs to perceive and understand his situation and be able to predict the outcome. Having a high level of SA is seen as one of the most critical aspects for achieving successful performance in aviation [1]. Many human errors in aviation are caused by lack of SA [2]. An adaptive pilot-plane interface can improve the flow of information between the pilot and the aircraft in such a way that the pilot's SA is improved and his mental workload is reduced. Such an interface can take advantage of a human's capacity for parallel processing via multiple sensory modalities such as vision, sound and feeling. As a result the survivability of the pilot and plane and the effectiveness of the mission will be improved.

1.2 Project description

The goal of the ACE project is "the definition and evaluation of a prototype adaptable interface technique to identify the specific automation requirements and practical utility of this innovation in a military cockpit" [3]. The idea is that certain features in the pilot-plane interface can be adapted or automated depending on the workload and status of the fighter pilot. Physiological measurements are taken to determine this workload and information from the aircraft system is used to determine the current situation. The high-level information supplied by the workload and situation assessment modules is then used by one or more agents (logical component) to determine the content, format and modality of the display in the cockpit.

In order for the ACE system to know where, when and how to perform adaptations the current situation is very imporant. This report deals with situation recognition within the aircraft using available aircraft data. This data needs to be abstracted to a higher-level goal (see fig. 1.1). Based on the goal the ACE system will determine in which area to adapt. The two logical methods to determine this context information are expert systems and neural networks. In this report we will look at the neural networks method.

1.3 Neural networks

At regular intervals we have recorded the values of certain pilot-control parameters (e.g. `throttle, brake, elevator`) and aircraft parameters (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

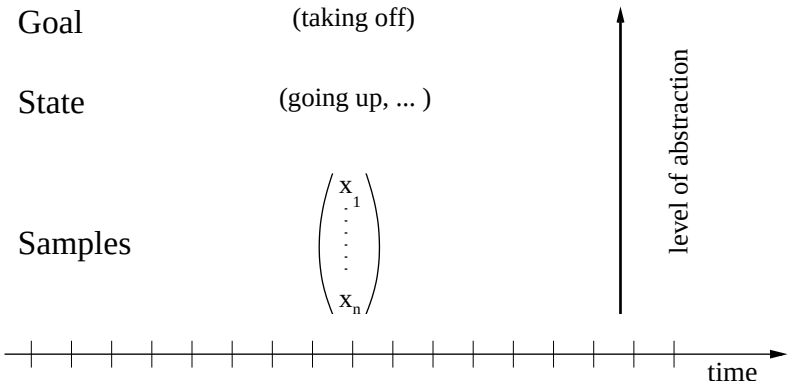

Figure 1.1: Illustration of levels of abstraction of the problem

is his goal. As a proof of concept we limit ourselves to the following set of actions: `going up, regular flight, turning right, turning left, going down, standing (on the ground), taxiing.`

We present two 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),

Chapter 2

Data Acquisition

To train any neural network, data is needed. In our case we needed realistic flight data from an aircraft. Currently, there are many advanced flight simulator software tools available. However, for the purpose described in the chapter 1, we need to manipulate input data and adapt the cockpit environment. For this reason we need the source of the software (also for the possibility of logging variables). Commercial available tools do not provide the source so we decided to use the free open-source FlightGear flight simulator [4].

2.1 FlightGear

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 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. The Flight Gear project is striving to fill these gaps [4].

Flightgear allows us to log and alter various aircraft properties and controls used by the pilot.

Figure 2.1: View from the cockpit in FlightGear

2.2 Getting the data

Settings for logging must be specified in the `preferences.xml` file in a `<logging>` section. The *sample rate* must also be set in this section. We used a sampling interval of *1 sec.* While flying a session with the flight simulator, the logged variables are written to a text file (name is specified in `preferences.xml` file).

Datasets were created in two ways: using the *autopilot* feature or flying manually. We used the autopilot because of its smoother courses of variables. Non-autopilot flights were done only by one person because it is not so easy to handle flying. The simulator is fairly realistic and it took us quite some time to learn to fly since none of us is a skilled pilot. We took many flights, about 40 with average length 15 minutes. We decided to use only good and successful flights (without crash and seeming to be similar to real pilot's flight).

A general problem with the datasets is that we do not know the right flying behaviour (as professional pilot do). At the moment, it makes no sense to generate rules (induced by these data) and use an expert system approach, since we do not have enough knowledge about the right flying behaviour. That is the reason we use neural networks to analyse this data. The results are of course dependent of training data, but the methodology is not, so we can use neural networks as a proof of concept.

Chapter 3

Data Analysis

3.1 Introduction

In this chapter we will discuss our 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 the *autopilot* feature, and sometimes they are also normalized or exponentially smoothed.

Some properties of the analyzed data we got by PCA analysis:

- eigenvalues of the covariation matrix of the data:

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

- eigenvectors of the covariation matrix of the data:

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

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 eigenvalues. We can see that the `pitch-deg` parameter has a large impact on p_1, which is influenced a little also by the `throttle` and `acceleration` parameters. The `acceleration` parameter has a large impact on p_2, which is also influenced by the `throttle` and `pitch-deg` parameters. The impact of the `roll-deg` parameter is rather small and that is the reason why `turning left` and `turning right` cannot be accurately recognized from the PCA figure in fig. 3.2. As expected these variables are more or less independent.

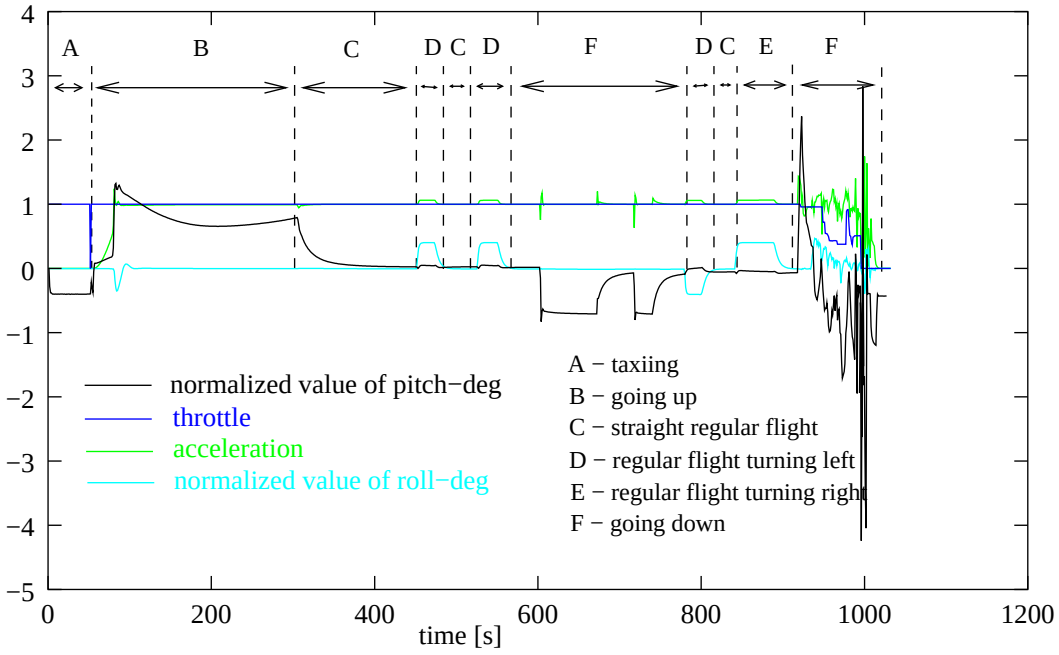

Figure 3.1: Time graph of selected flight variables during a simple flight (using the autopilot feature).

3.2 Basic view of the data

We started with a selection of four logged variables that are shown below. 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 did not make any precise analysis (reason for this will be described later). In fig. 3.1 the time graph of the flight data is shown. Fig. 3.2 shows the PCA and other projections of data acquired using the autopilot feature that was used to detect separate clusters. Fig. 3.3 shows the projection of the variables' tracks during a simple flight. It also shows that is is possible to follow the stages of a flight by following the tracks made by the logged variables. The left part of the figure stands for approximately the first half of the flight and the right one for the rest of the flight. For this analysis, the data was exponentially smoothed, $\alpha = 0.03$.

The variable `roll-deg` obviously 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 in the figures. The variables `throttle` and `acceleration` are more complex, but they are able to recognize the rest of the states, as can be seen in the figure.

In the figures 3.2 and 3.3 it can be seen that there can be a problem with data classification: the clusters lay over one another. This can be caused by the data logging or maybe it is just a problem of a complicated interference between variables.

3.3 Differenced cross-correlation functions

In this section we look at some relations among the various variables that we logged. We consider the differenced cross-correlation to be more important than regular cross-correlation because we are

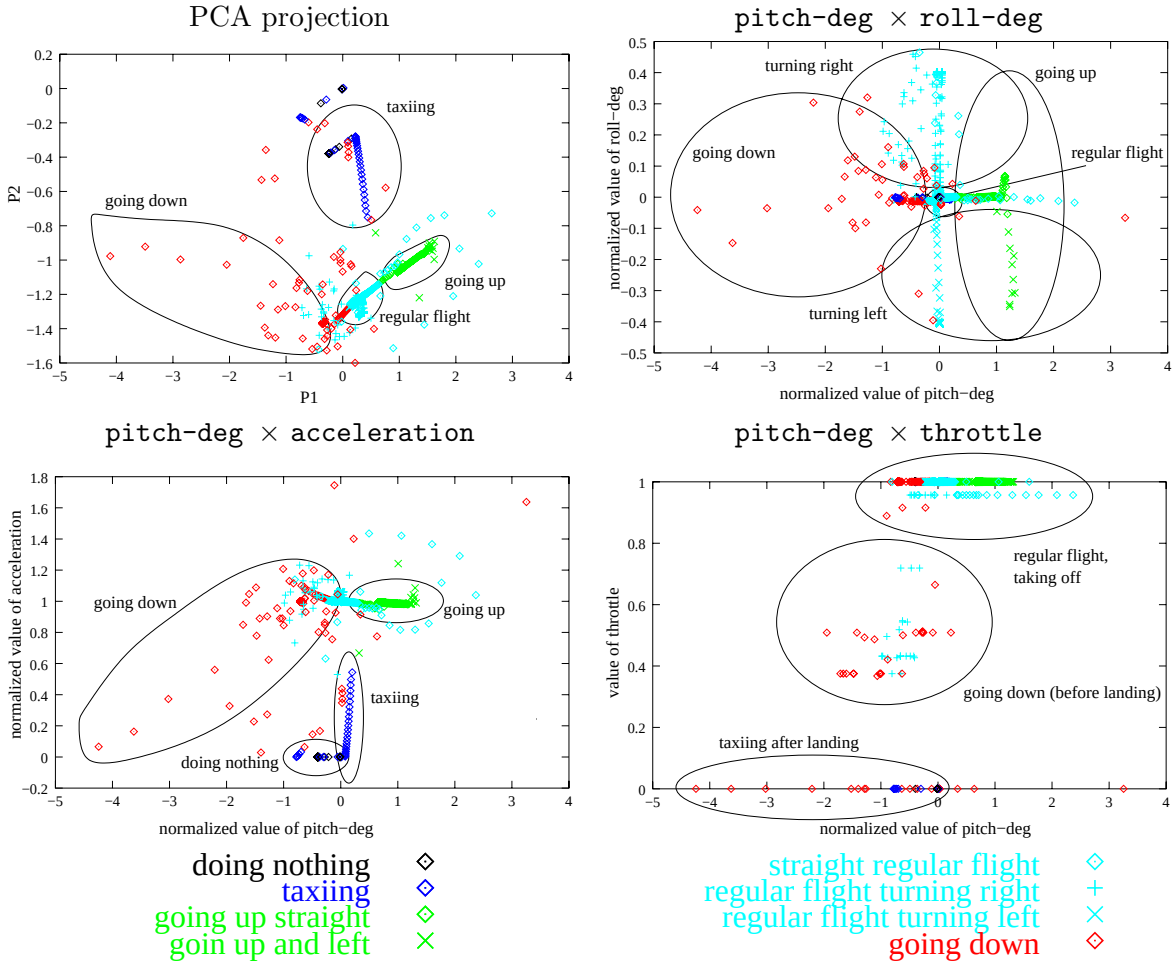

Figure 3.2: Various projections of flight data

mainly interested in trends. Also the course of regular cross-correlation is much worse (we did not put a figure here); it does not 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 and 3.5 show the cross-correlation functions of the variables. These cross-correlation functions are made of non-normalized data (whole flight). Cross-correlation can be computed 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.

In fig. 3.4 you can see that `acceleration` with `elevator`, `acceleration` with `pitch-deg` etc. are strongly correlated, especially when compared to fig 3.5 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) dependence among most important variables can be seen. For example, the `acceleration` / `elevator` cross-correlation function has a maximum in the lag equal to zero, which means that the `acceleration` and `elevator` parameters are correlated and the dependency is not delayed. On the other hand, the `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 comparison with `roll-deg` parameter). From cross-correlation functions courses (fig. 3.5) 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 defined confidence limits.

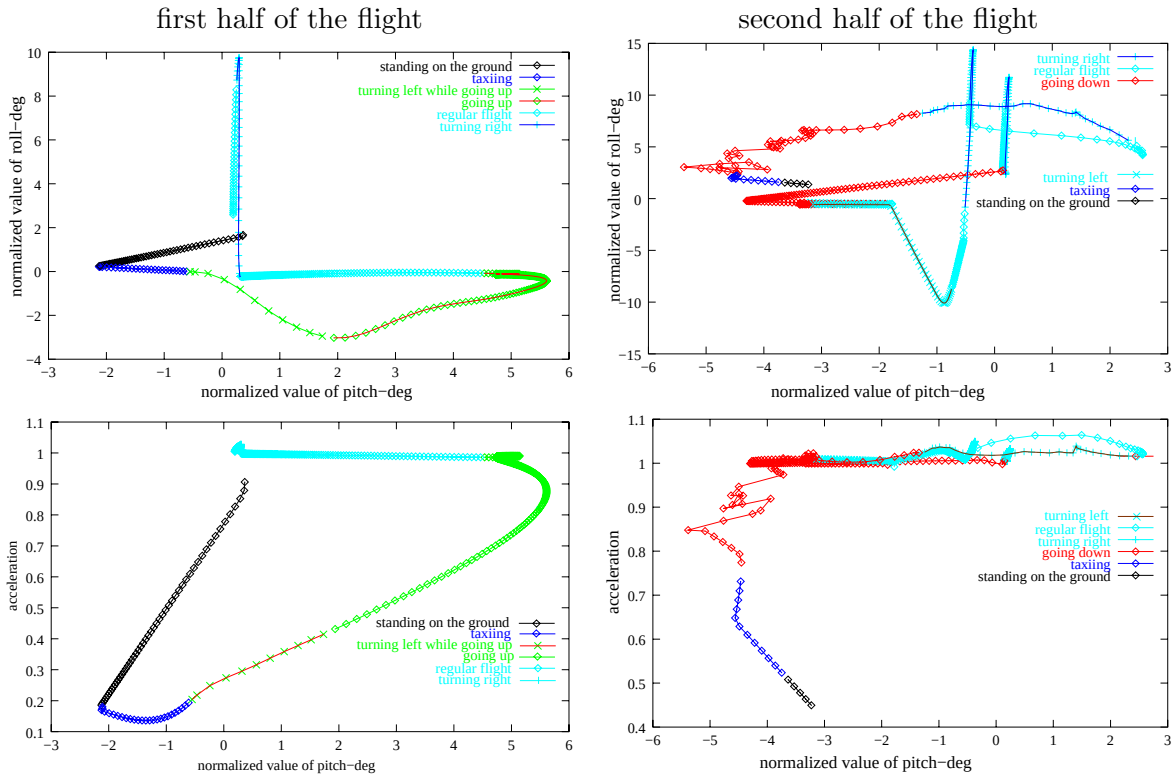

Figure 3.3: Courses of parameters of the flight – smoothed data

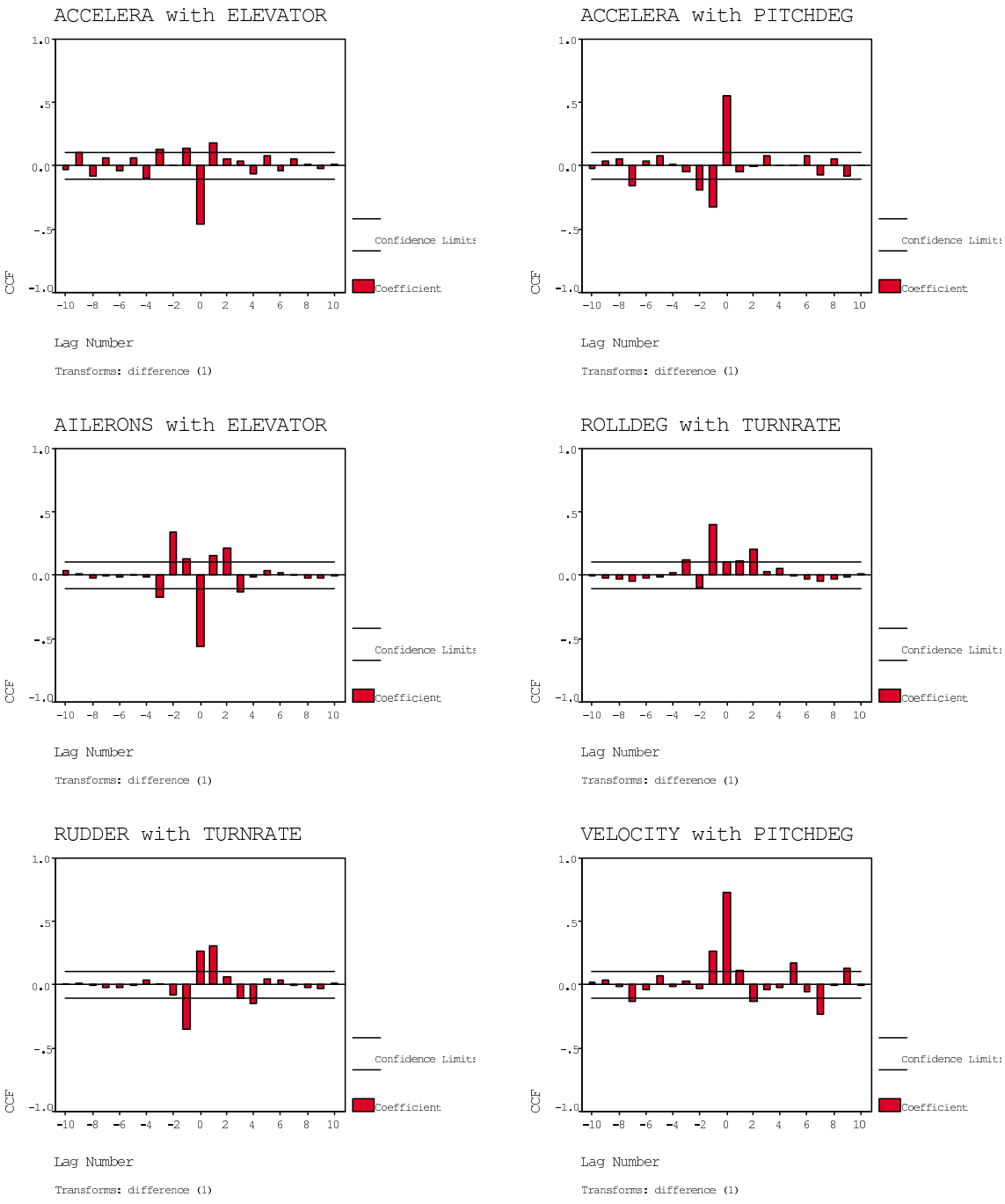

Figure 3.4: Differenced cross-correlation functions of related variables

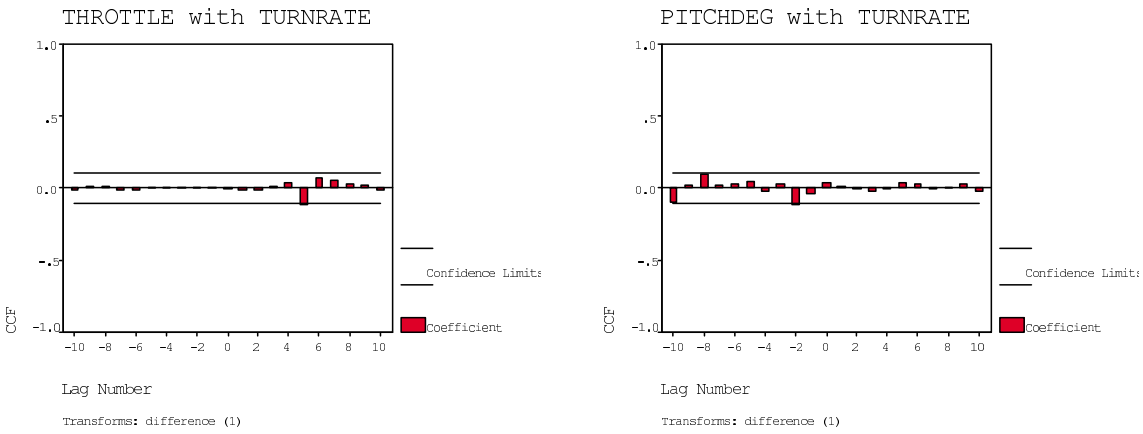

Figure 3.5: Differenced cross-corelations functions of unrelated variables

Chapter 4

Assessment of the Activity/Action and Goal of the Pilot

4.1 Approach

Our goal is to explore if it is possible to use artificial neural networks (ANNs) for the assessment of action/situation. We decided to discriminate only among seven basic situations (described below). We used a (partly) recurrent (*Elman*) network because a time context exists among situations. We generated the data using the *FlightGear* flight simulator that was dicussed earlier. For handling ANNs we used the *Stutgart Neural Network Simulator (SNNS), version 4.2* (See [5]).

For training our neural network we of course need data that is already classified. To be able to attribute the desired network output to the input data, we manually logged the pilot's actions over time. A problem with logging data was that although we set the *sample rate* to *1 sec* , the variables were logged over periods of about *1.2 sec* by the simulator. Handling of this problem will be described later.

4.2 Data preprocessing

The structure of the data file acquired from the Flightgear simulator (`.log` file) is different from the pattern file (`.pat` file) that is read by SNNS. Therefore, we created a program for data conversion and preprocessing. The program also places the desired outputs of the network into the `.pat` file. For every flight we created another file, named `x.act` (where `x` stands for the name of the logfile) to store the codes and durations of various epochs of our flight. Our program then writes these patterns to the `.pat` file. The structure of the `.act` file will be described below.

Training of a neural network can sometimes be a problem if the input variables are from different ranges, so our program also provides simple normalization. This means you can divide a variable by any desired float number. If normalization is wanted then the current directory must include the file `x.nrm`, where `x` stands for the name of the logfile without extension again. The structure of this file will also be described below.

4.2.1 Structure of the `.act` file

The sign '%' stands for a comment continuing till the end of the line. The first number in the file contains the *total time* of the flight in seconds or minutes. This approach solves the problem of the sample rate as described above by computing the ratio of the time of this action to the total time. The second number in the file is the number of outputs, which is recommended to be the same as the number of recognized actions. Next follows the desired outputs of the neural network for each action and behind the multicollon `";"` 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 is no rule about

repetition of each action but if these data do not correspond to the `.log` file, the network would be trained badly. An 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
```

4.2.2 Structure of the `.nrm` file

The first line stands for the number of variables which should be normalized. The other lines contain the normalization factor first and then the name of the variable. This name must correspond to that in the `.log` file. 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
```

4.2.3 Function of thr `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 is 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 level (not going up or down) and turn left at the same time, or turning while going up. The coding was chosen because we thought that it would be 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

The 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 actual 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 [7]. An example of an Elman network with one hidden layer is in fig. 4.1.

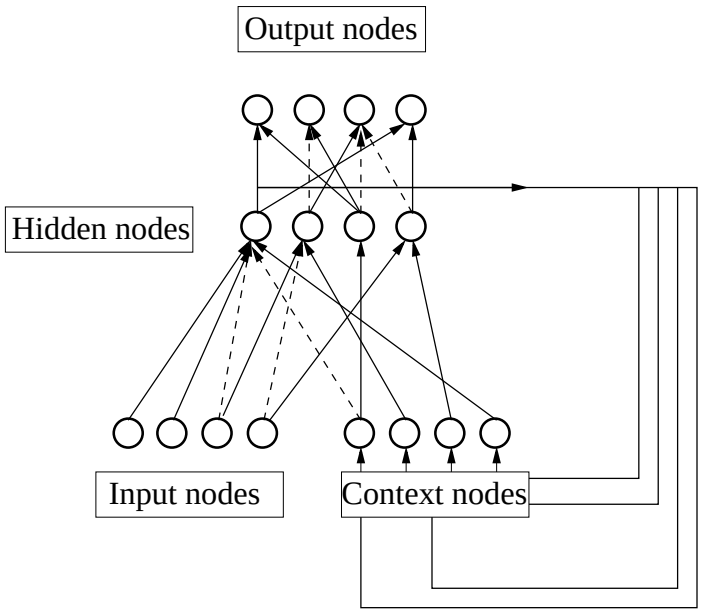

Figure 4.1: Simple Elman network – 4 input units, 1 hidden layer with 4 units, 4 output units, 4 context units

4.5 Experiments

Several experiments were done to find a good network structure and settings, as well as the best variables to use. Below we describe some of our findings.

Experiment 1:

In our first experiment we tried to create some data and load them into a neural network. We logged 17 variables, did not 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 only the Elman network with output context.

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.

Experiment 2:

In experiment 2 we decided to normalize our data. One approach 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 should be run in real-time in the future. It is also unusable to normalize each data set to different mean and different variance.

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. From this experiment we concluded that that output context seems to be important as well as 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`. We tried using `vertical-speed` instead of `pitch-deg`, but `pitch-deg` is much more smoothed. The problem of PCA is that it emphasizes the variables with the largest variance, not considering the importance of the changes.

Experiment 4:
We still had problems the with `throttle` variable. Every flight used different levels of throttle, so we added a more objective variable – `acceleration`. For better precision in the recognition of `standing` and `taxiing` we left the `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 provided very good results, even on the data from non-autopilot flights.

4.6 The neural network

We decided to use an Elman neural network with 2 hidden layers, 10 units each and output context. 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 used 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 did not change during learning) and $\psi = 0$ which is the initialization value of context units. For more details, see [5]. For learning, we used regular back-propagation, with the step width $\eta = 0.2$. The error on which learning 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 used the regular `JE_Order` function.

4.7 Results

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

Figure 4.7 shows the training and testing error of the network, trained on the data acquired with the 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 sum up some of the badly classified patterns (we consider the significantly highest value of all the outputs):

serious mistakes – 1.4 %, e.g. turning left when the pilot did not turn left at all etc.
less important mistakes – delayed performance, mean delay 17.0 patterns, standard deviation 7.6 patterns.

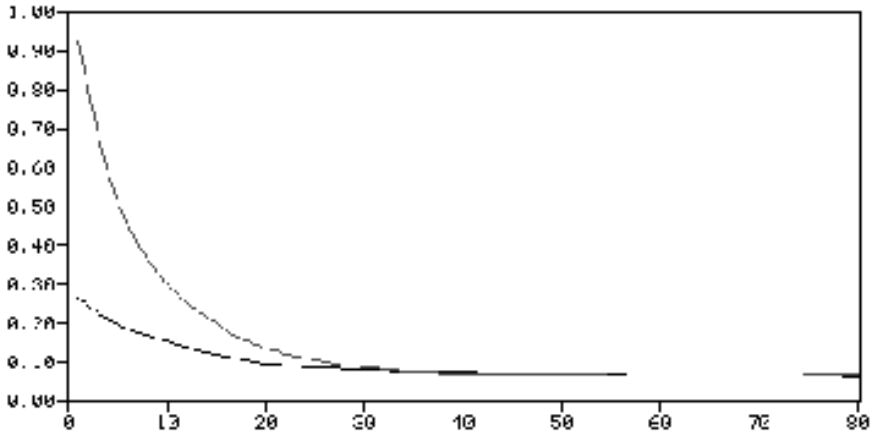

Figure 4.2: Training (lower) and testing (upper) mean square error (on different data)

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

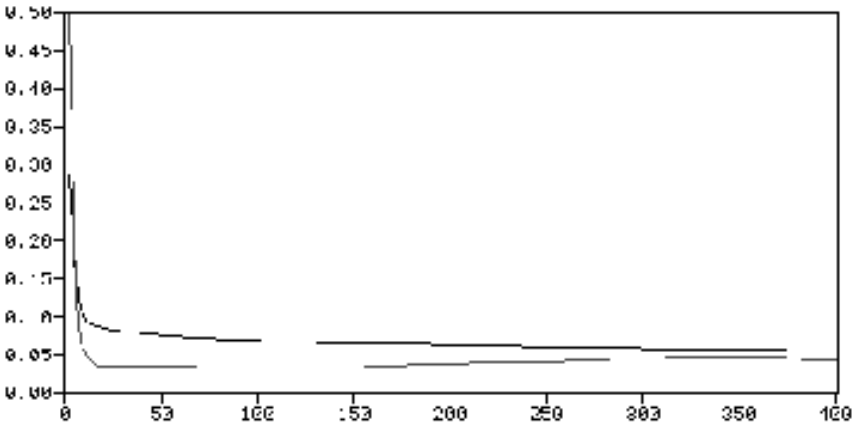

Figure 4.3: Training (higher at the end) and testing mean square error (simple manual flight)

4.8 Conclusions

It is possible to recognize the state of the aircraft (e.g. going up, going down) from the variables we logged with the use of an recurrent ANN. The problem of recognizing the goal of the pilot is more philosophical – it is usually correlated with the state of the aircraft but not necessarily. Nevertheless, when recognizing the state of the aircraft we needed to put a priori information into our classifier but we still think it was neccessary. Training of the neural networks was complicated and took a long time. We suppose that some logging sessions were less "useful" than others. Also the normalization factors are given a priori instead of calculated, but they are used just for shifting the input data into a reasonable range.

Chapter 5

Predictions of Flight Variables Using Neural Networks

5.1 Introduction

In this chapter we will discuss our approach to predict future values of the logged parameters. For this, we need to know the previous values of these parameters. We decided to use ANNs once more, just feedforward ones this time. However, many variations are possible:

- the size of the **history time window**, which means how many past values will be considered, can be chosen (see also 5.1);
- the size of the **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 from which flight parameters the predictions should be made.

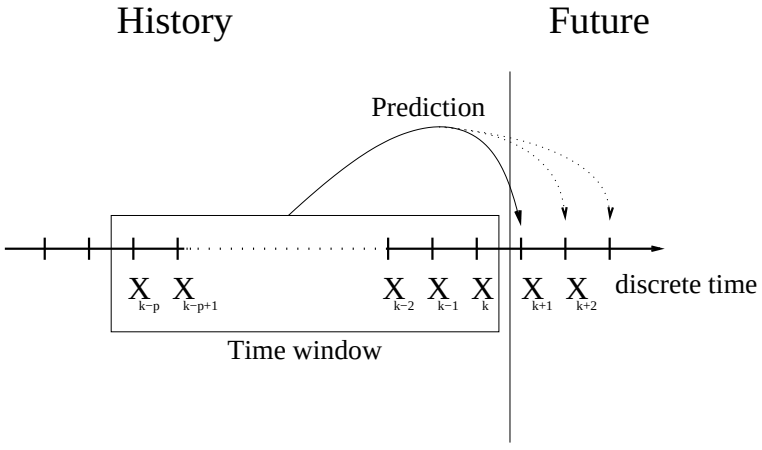

Figure 5.1: Time window that is used to predict future values

We have used a regular feedforward network with standard back-propagation learning method. Parameters of learning will be mentioned later. For training the network we used the same data as shown in fig. 3.1. For testing, we used other data, but all generated with the use of the *autopilot* feature.

5.2 Data preprocessing

For data preprocessing, we devised a program similar to the one used for the approach described in the last section. This new program needs a `.nrm` file and of course also the `.log` data file as well as several additional parameters. The first parameter is the name of 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), 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). For example:

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

Here the program will take the file `data.log` and `data.nrm` (if desired) and puts the results into the `data.pat`file. The input variables of the network will be 10 past values of the second and third variable logged in the `.log` file (note 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 (see fig.3.1). The sampling rate was set to 1 sec, which means slightly more that 1000 patterns. We planned several experiments with various sizes of time window. We used a regular feed-forward network with standard back-propagation and step size 0.2 (mostly). In most cases learning was done 1000 epochs and no overtraining appeared. Weights were initialized randomly in all cases, in the range of $\langle -1, +1 \rangle$. Both training and testing data were normalized, with the same factors used, in all cases. The output layer and the last hidden layer used a pure linear activation function to provide the continuous outputs. A sigmoidal function was used in all other cases.

5.4 Results

Below we have summarized the results from several experiments.

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 training error, 0.01026 testing error (more details in fig. 5.2).

Acceleration: MSE 0.02847 training error, 0.00564 testing error (more details in fig. 5.3).

Roll-deg: MSE 0.00115 training error, 0.00029 testing error (more details on fig. 5.4).

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 training error, 0.12102 testing error (more details in fig. 5.5).

Acceleration: MSE 0.00454 training error, 0.00995 testing error (more details in fig. 5.6).

Roll-deg: MSE 0.00538 training error, 0.00791 testing error (more details in fig. 5.7).

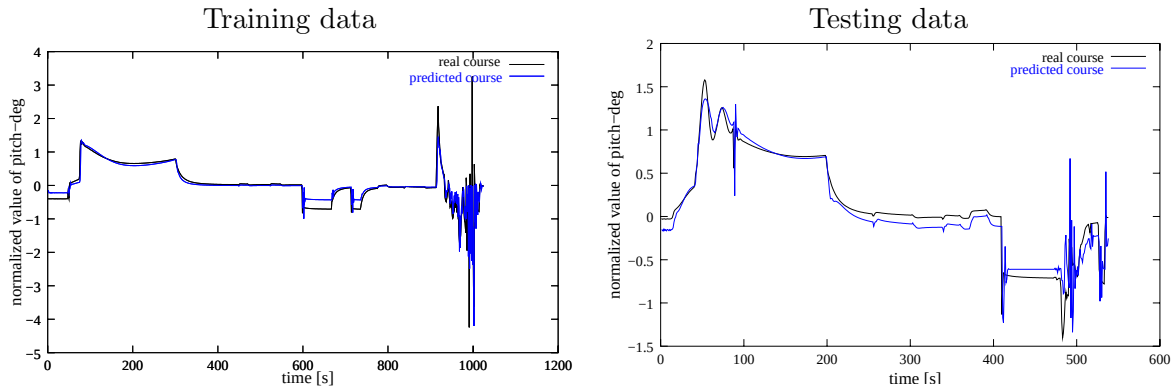

Figure 5.2: Prediction of the pitch-deg variable, time window size 5

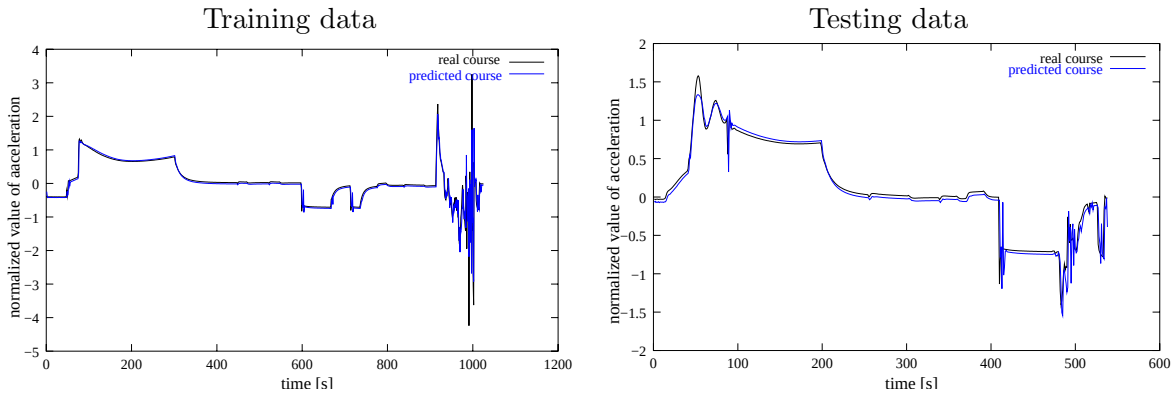

Figure 5.3: Prediction of the acceleration variable, time window size 5

5.4.3 One variable one-step prediction from all variables

Network description: 2 hidden layers, 10 units in the first and 5 units in the second layer, time window size 5. All 4 logged variables were used for prediction, i.e. 20 input units.

Pitch-deg: MSE 0.00329 training error, 0.11834 testing error (more details in fig. 5.8). (Step size learning parameter decreased to 0.1 for technical problems.)

Acceleration: MSE 0.00170 training error, 0.02623 testing error (more details in fig. 5.9).

Roll-deg: MSE 0.00047 training error, 0.00760 testing error (more details in 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 and 4 output units. Step size 0.1, 1500 epochs (technical problems occured with divergence of the network)

MSE: 0.01334 training error, 0.09575 testing error (more details in figure 5.11).

5.4.5 More steps ahead prediction of the `pitch-deg` variable

Network description: 2 hidden layers, 10 units in the first layer and 8 units the second layer, 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 layer and 5 future values of the variables are the teaching output of the network. Particular

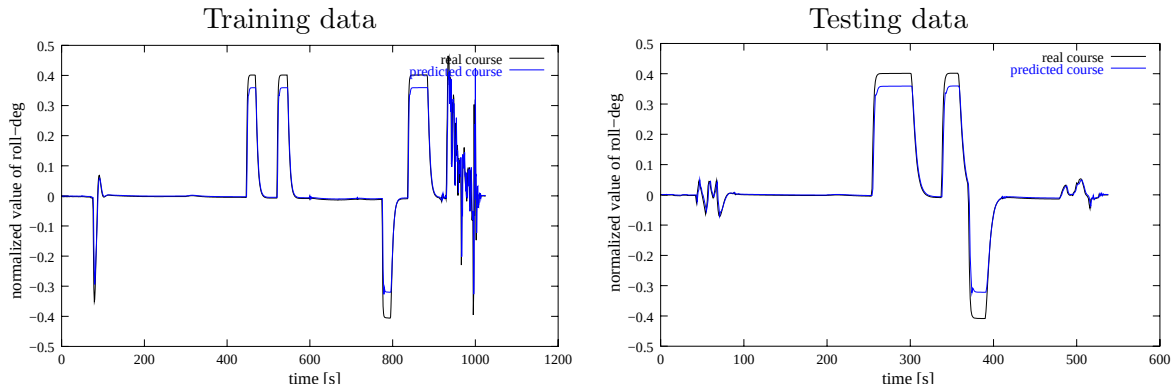

Figure 5.4: Prediction of the roll-deg variable, time window size 5

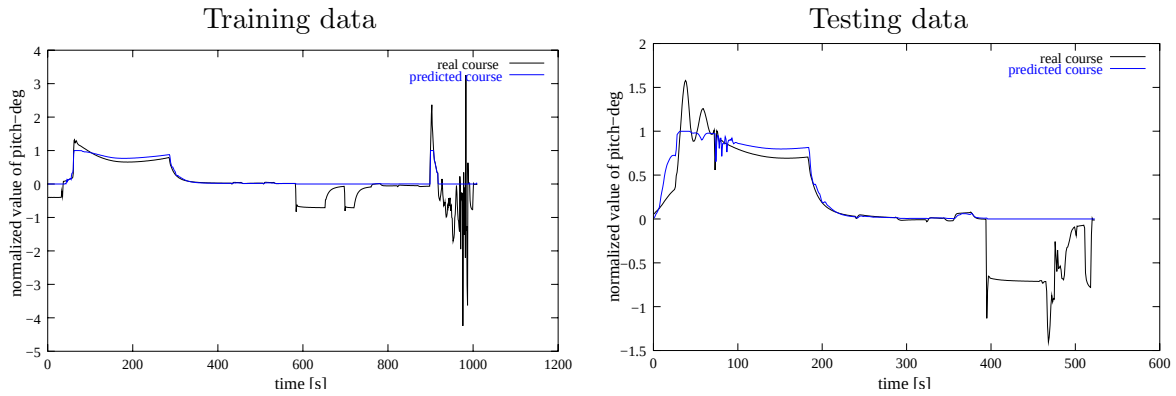

Figure 5.5: Prediction of the pitch-deg variable, time window size 20

prediction performance is shown in figures 5.12, 5.13, 5.14.

MSE: 0.19639 training error, 0.62649 testing error
(learning parameters: 7000 epochs, 0.05 step size)

5.5 Concluding remarks

The one variable one-step prediction with the history time window size of 5 seems to be fairly successful, except for bad levels of the signal. Also, the timing is right. We consider these results to be quite good. Large history time window has a poor performance, which can be attributed to the bigger complexity of the neural network and more difficult learning. We think that the network has a problem recognizing the "importance" of each of the data values. One-step prediction of one variable from all variables is somewhat disappointing, because we supposed that more information would result in better predictions. The problem of the poor performance of one variable, one step from just one variable can be attributed to the complexity of the network. Perhaps longer learning times or using more data will improve performance. The same applies to the prediction of all variables from all variables. More-step prediction performance corresponds to our expectations. The prediction error is getting bigger with the number of steps predicted ahead. A problem in this case is the great difference between performance of training and testing data, but this can probably be reduced by using more training data.

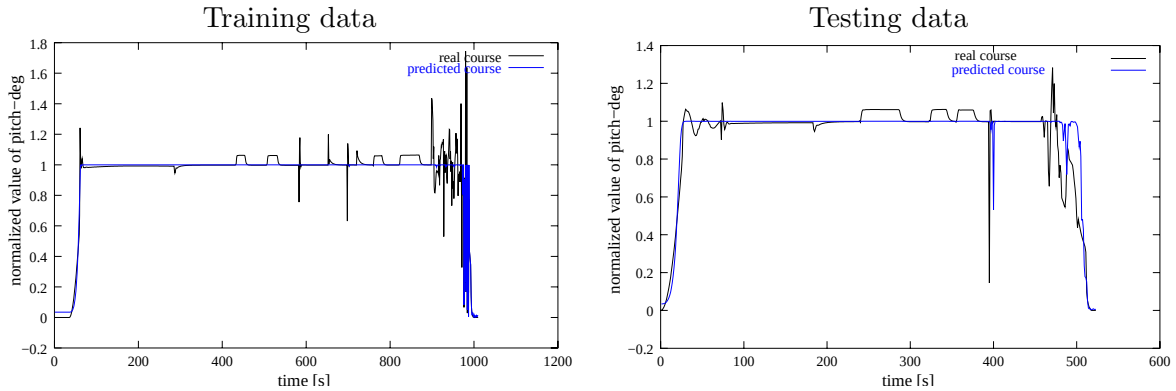

Figure 5.6: Prediction of the acceleration variable, time window size 20

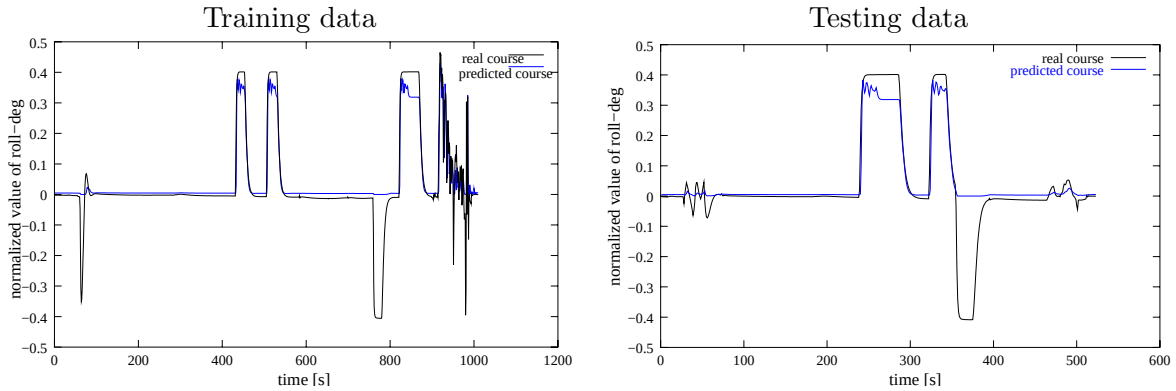

Figure 5.7: Prediction of the roll-deg variable, time window size 20

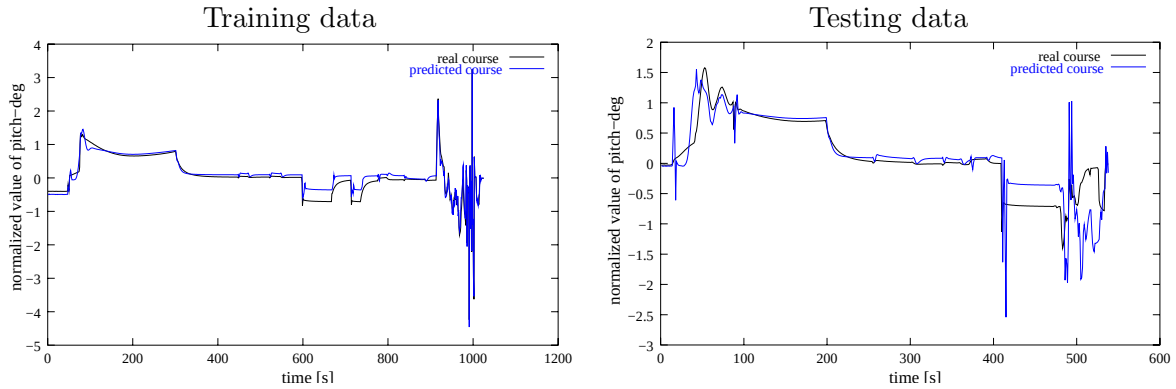

Figure 5.8: Prediction of the pitch-deg variable from all variables, time window size 5

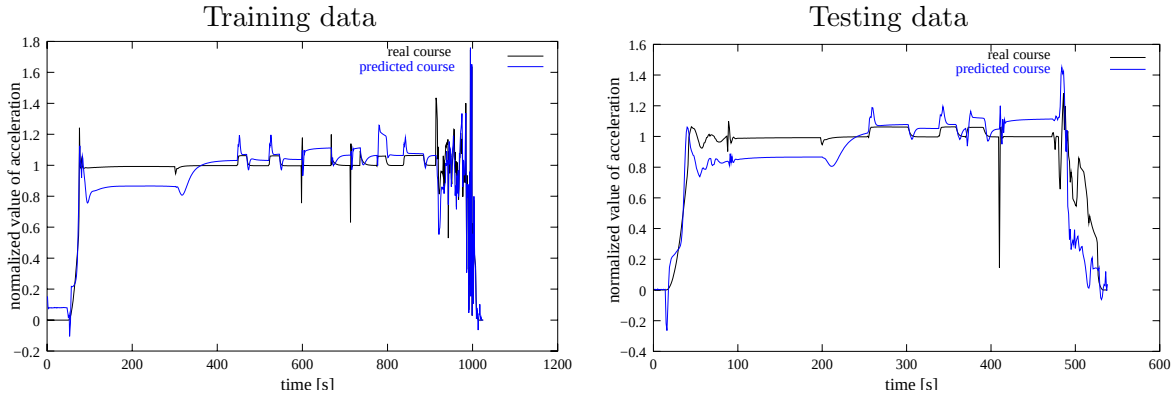

Figure 5.9: Prediction of the acceleration variable from all variables, time window size 5

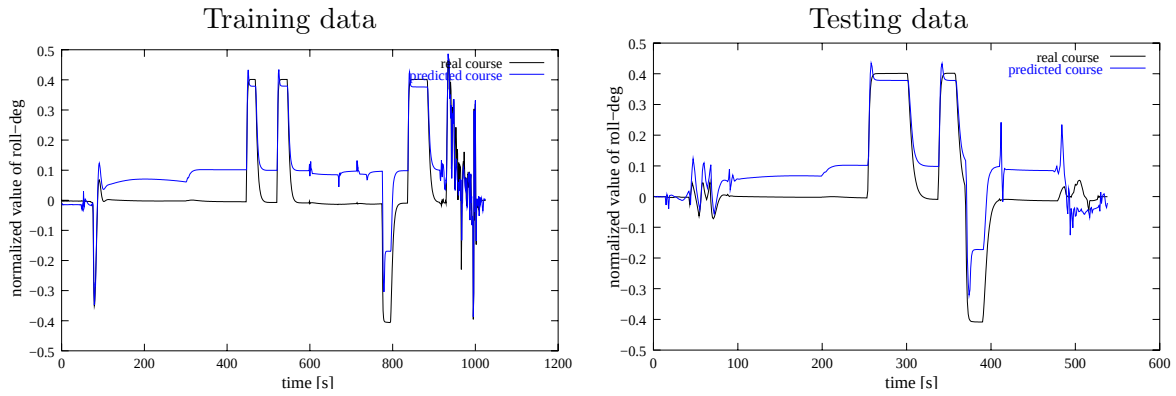

Figure 5.10: Prediction of the roll-deg variable from all variables, time window size 5

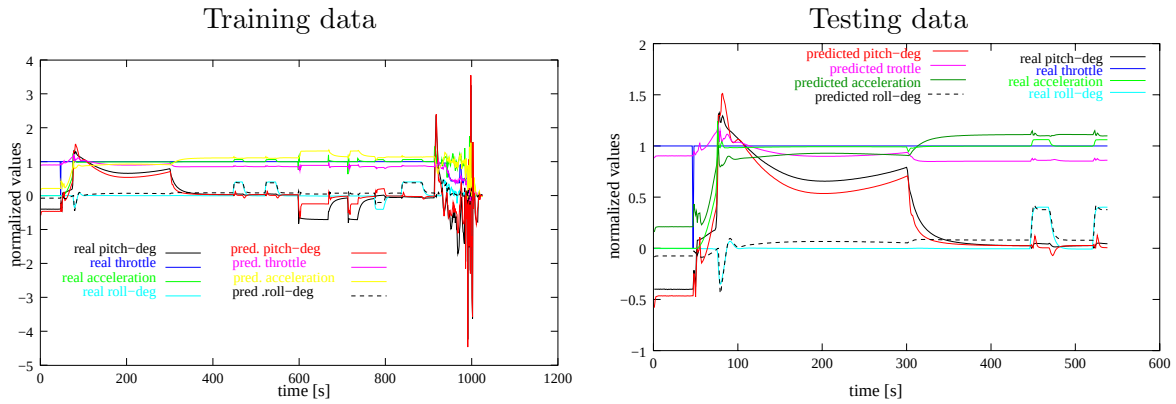

Figure 5.11: Prediction of all variables from all variables, time window size 5

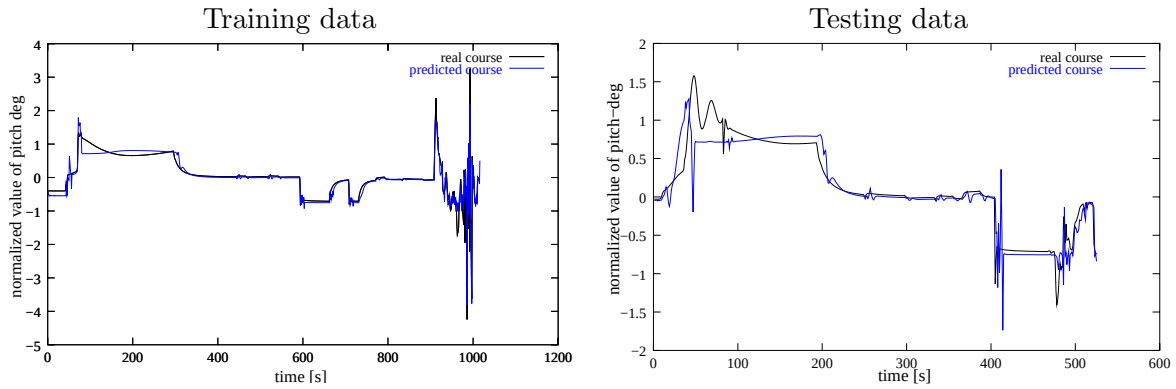

Figure 5.12: 1-step ahead prediction performance (`pitch-deg`)

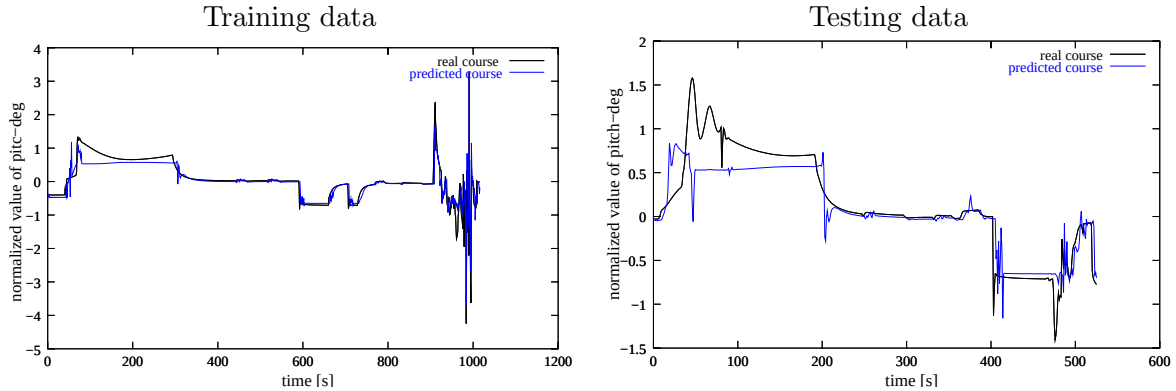

Figure 5.13: 3-steps ahead prediction performance (`pitch-deg`)

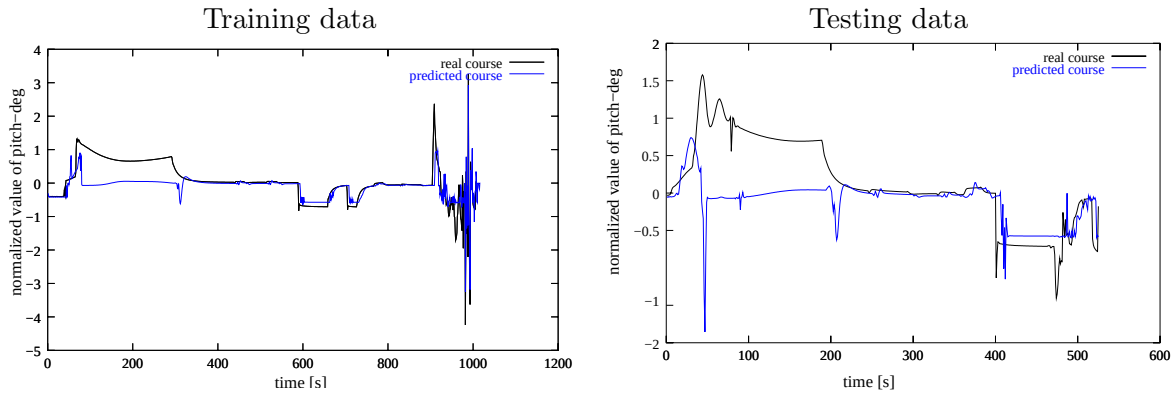

Figure 5.14: 5-steps ahead prediction performance (`pitch-deg`)

Chapter 6

Conclusions and Future Work

The goal of this work was twofold. First we wanted to explore and "get a feel" of the data from a flight simulator. Second, we wanted to test how artificial neural networks (ANNs) would perform in providing the necessary context information for future intelligent pilot- vehicle interfaces. The main advantage of ANNs over expert systems is the fact that they require little or no domain knowledge. A drawback is that errors can be made by ANNs, making the result not always reliable. Since a pilot always has to be able to rely on the given information and the accuracy of the system, we recommend not to use neural networks as the main information source for an intelligent pilot-vehicle interface system. We feel that an expert system would be much better and more predictable in this case. However, neural networks can be used as an additional or extra information source for pilot action estimation in case of uncertainty. Also, for predicting future values of variables ANNs can be used. The latter does require a careful setting of the size of the time-window that is used to obtain the prediction.

This report stands at the beginning of the Adaptive Cockpit Environment project. Its purpose is just to sketch which approaches are possible to use and how successful neural networks might be. A lot of work needs to be done in the future. One of the first activities, (which we have already started) should be to acquire the necessary background domain knowledge for creating an expert system. It should also be investigated what further role nueral networks can play in combination with an expert system.

Bibliography

[1] Endsley, M.R. (1999a) "Situation awareness in aviation systems", Handbook of Aviation Human Factors, Garland, D.J., Wise J.A., Hopkin, V.D. editors, Lawrence Erlbaum Associates, Mahwah, NJ USA.

[2] Endsley, M.R. (1999b) "Situation awareness and human error: designing to suppport human performance", Proceedings of the High Consequence Systems Surety Conference, Alburquerque, NM USA.

[3] NLR (2000) "Adaptive Cockpit Environment", memorandum VE-2000-002, Version 1.1, Nationaal Lucht-en Ruimtevaartlaboratorium, The Netherlands.

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

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

[6] Mulgund, S. S. and Zacharias, G. L. (1996) "A Situation-Driven Adaptive Pilot/Vehicle Interface", www.cra.com/publications/papers/hics96.pdf

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

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

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

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

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