RECOGNISING SITUATIONS IN A FLIGHT SIMULATOR ENVIRONMENT

Patrick A.M. Ehlert, Quint M. Mouthaan *and* ***Leon J.M. Rothkrantz***
Data and Knowledge Systems Group
Department of Information Technology and Systems
Delft University of Technology
Mekelweg 4, 2628 CD Delft, the Netherlands

E-mail: P.A.M.Ehlert@its.tudelft.nl, L.J.M.Rothkrantz@cs.tudelft.nl

KEYWORDS

Artificial intelligence, flight simulator, context awareness, A.I. bot, neural networks, knowledge-based system

ABSTRACT

In this paper we describe our approach to a situation recogniser system that currently is being developed for a flight simulator environment. The situation recogniser is part of a context-aware system and can be seen as a first step to an artificial intelligent pilot bot. We will address our explorative data study (PCA analysis), our attempt to recognise and predict situations with an Elman neural network, and our choice to use a knowledge-based production system.

INTRODUCTION

Ever since the first airplane was built by the Wright brothers the capabilities of aircraft have continuously been improved. For example, the maximum speed of the average military fighter plane has gone from approximately 100 Mph in 1920 to over 1500 Mph currently. These high speeds are responsible for the little time available to pilots to process information and make decisions. In addition, the improved range of weapons in military aircraft (missiles can be fired from 20 km away) reduces the pilot's decision time even more. Also, the amount of information available to a pilot today and the complexity of the contents have increased significantly. Where earlier planes only had a few meters, modern aircraft have several hundreds of meters or information displays, providing the pilot with a wealth of different information sources.

To help the pilot deal with information processing and decision-making, and to avoid cognitive overload, a crew assistant system or intelligent pilot-vehicle interface (PVI) has been proposed [Mulgund and Zacharias 1996]. The idea is that such a system would present relevant information to the pilot at the right moment, depending on the situation, the status of the aircraft, and the workload of the pilot.

The Data and Knowledge Systems group at the Delft University of Technology is currently working on a project called Intelligent Cockpit Environment, or ICE for short. The main objective of this project is to investigate new interface techniques and technology for intelligent PVIs. Part of the ICE project is to design a context-aware system that can automatically recognise the current situation of the pilot and aircraft. The first step towards this context-aware system is to create a situation recogniser module. The situation recogniser module should be able to determine the status of the aircraft and the corresponding phase in the flight plan.

Although the ICE project does not explicitly focus on creating an A.I. pilot bot capable of reasoning and recognizing situations in a flight simulator, it should be possible to use the context-aware system for these purposes.

THE FLIGHTGEAR SIMULATOR

Many sophisticated flight simulator software packages are available on the market, but most programs are commercial software that cannot be extended. For the purpose described above we want to be able to manipulate input data and adapt our cockpit environment. Therefore, we chose the open-source FlightGear flight simulator as our experiment platform (see also Figure 1).

Figure 1: Screen shot of the FlightGear program

The FlightGear simulator project is an open-source, multi-platform, cooperative flight simulator project. The idea for FlightGear was born out of dissatisfaction with current commercial available PC flight simulators. 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 simulations ideas, and an good and extendable end-user application [Perry and Olson 2001]. The FlightGear platform is open to be expanded and improved upon by anyone interested in contributing. For more information on FlightGear visit the website http://www.flightgear.org

EXPLORATIVE DATA ANALYSIS

We started our research with an explorative data analysis. The FlightGear simulator allows us to log almost all internal variables (e.g. altitude, airspeed, gear position, etc). For our explorative data analysis we selected four variables: pitch, throttle, acceleration and roll. Figure 2 shows the time graph of the flight data generated on a sample flight. Note that the straight flight (part C) was flown using the auto-pilot.

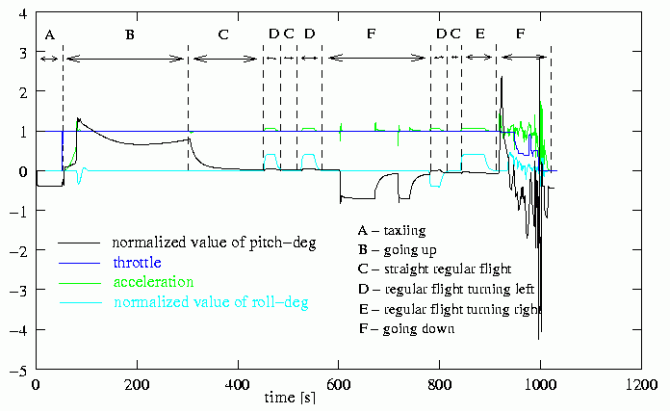

Figure 2: Time graph of selected flight variables during annotated sample flight

PCA Analysis

The goal of the PCA analysis was to investigate the possibility to give an automated interpretation of recorded data; what was the planned action of the pilot and what was his goal. As a proof of concept we limited ourselves to the following set of actions: going up, regular (straight) flight, turning right, turning left, going down, stand still (on the ground), and taxiing.

Applying principal component analysis (PCA) or Sammon mapping we were able to project the logged data and cluster the data in the 7 selected action states. Figure 3 shows two projections of variables' tracks during our sample flight. From this figure we conclude that in principle it should be possible to define states, which will result in distinct clusters in the space of logged data. By tracking the (projected) flight we can label the position with the corresponding label of the cluster. This way we are able to give an automated interpretation of the flight behaviour based the logged data as is shown in Figure 4.

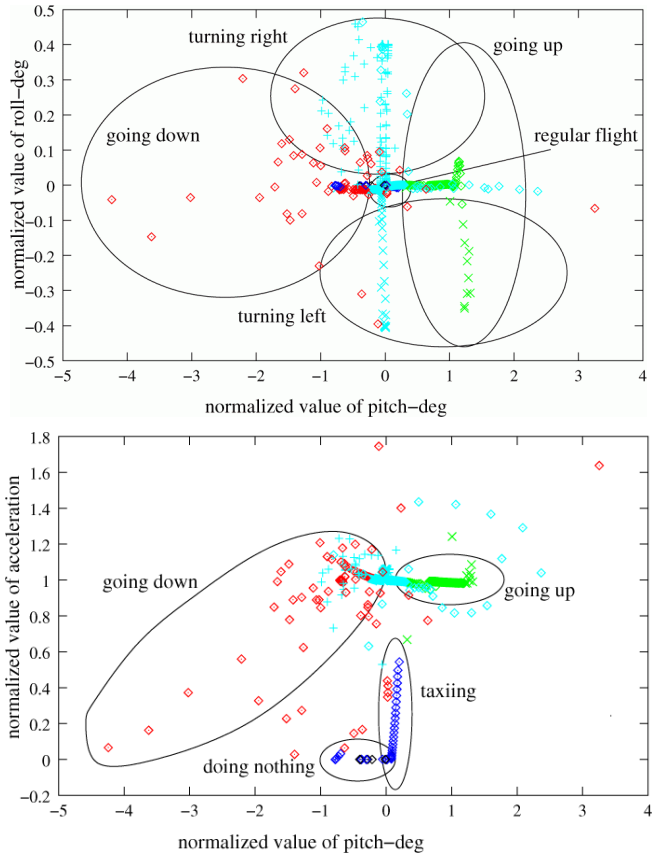

Figure 3: Clustering in two PCA projections

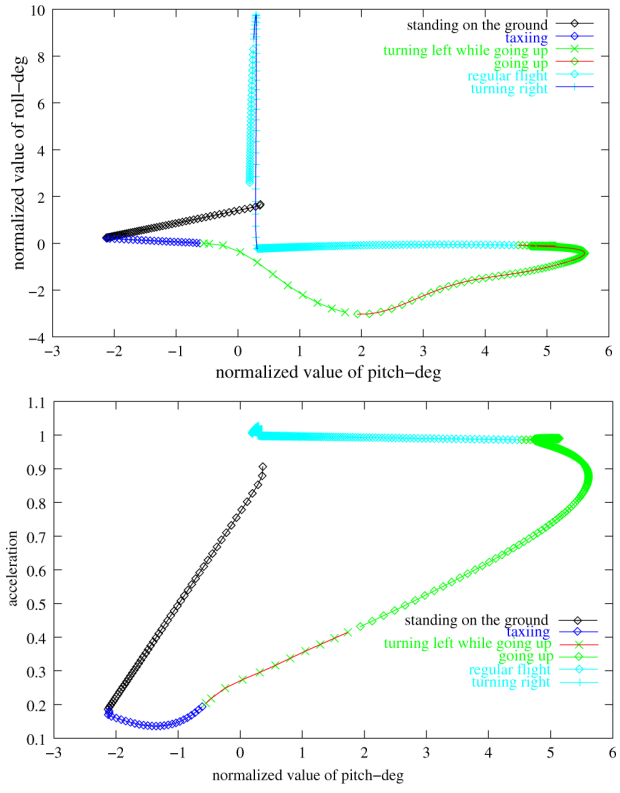

Figure 4: Tracking path in the two PCA projections

Elman neural network

We found similar results using recurrent neural networks. We selected an Elman neural network with one hidden layer as is shown in Figure 5. As test input we used the same logged data as before and as output the earlier-mentioned 7 states. We were able to train the neural network for the automatic recognition of the 7 states. The error rate on a set of test data was 13.5 %.

We also used neural networks to predict the future values of the logged parameters. As displayed in Figure 6, for every variable X, we used at every point k the previous values (X_k,…X_{k-p}) to predict X's future values (X_{k+1}, X_{k+2}). In Figure 7 we show the results using a feed forward network of two hidden layers (4-5-5-2 architecture) using window size 5.

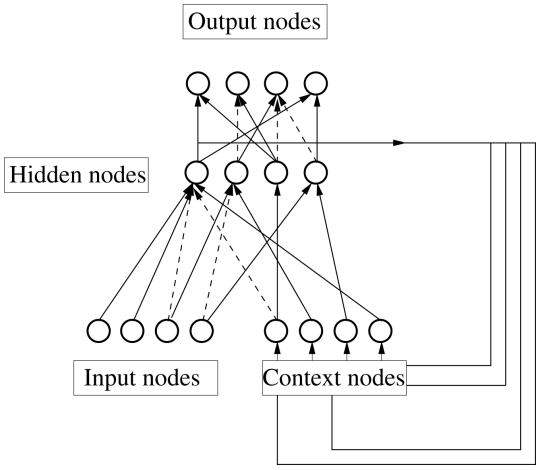

Figure 5: Architecture of used Elman neural network

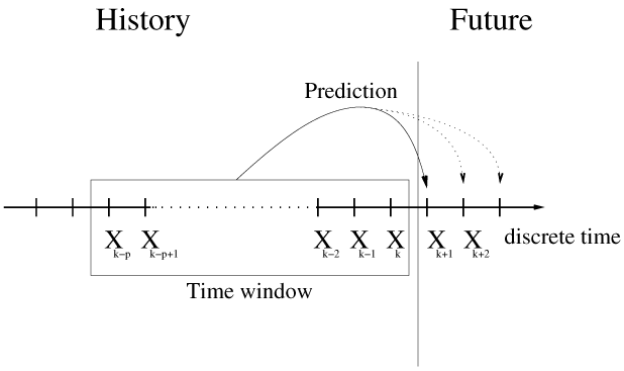

Figure 6: Model of prediction

More results about the PCA analysis and the prediction with Elman neural networks can be found in [Capkova, Juza and Zimmerman 2002].

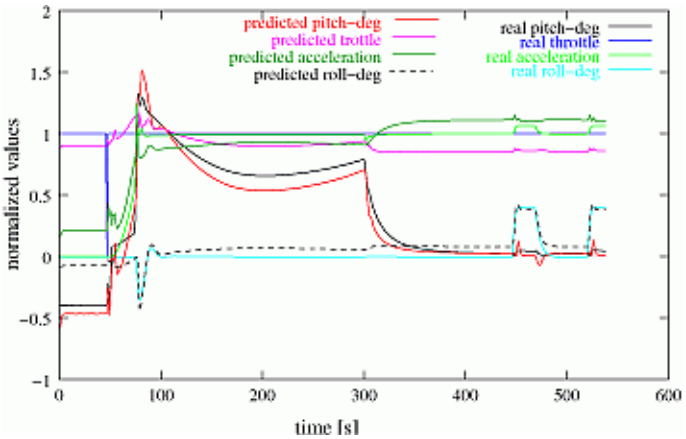

Figure 7: Results of neural network state prediction with window size 5

KNOWLEDGE BASE

After the explorative data study, we decided to take a knowledge-based production system as the basis for our real-time and on-line situation recogniser module. The advantage of a knowledge based system is that it is much more transparent how the system makes a decision, compared to the neural network approach. In addition, it is possible to make changes to the knowledge base and adapt the system to new circumstances or environments (e.g. other aircrafts).

A simple prototype

For our prototype program, we started with designing a set of rules to recognise situations that can occur while flying a Cessna 172, the default airplane in FlightGear. We made rules for the following situations; start-up, taxiing, hold-short, take-off, aborted take-off, set course to waypoint, in flight, start-landing, aborted landing, final approach, touchdown and shutdown. All situations can be recognised based on a number of parameters such as airspeed, vertical speed, throttle, brakes status, gear status, etc. For each state we tried to use as much of the available variables as possible, since this allows us to still get an accurate indication of the situation, even if one of the parameters is not normal for that situation. For example, if the pilot lowers the gear, it is obvious that he is trying to land. However, if for some reason the pilot forgets to lower the gear, we are still able to determine that the pilot is landing by looking at his airspeed, flaps, vertical speed and altitude. This allows us to provide feedback to the pilot about possible mistakes or malfunctions in a latter stage.

To reduce the amount of rules that have to be checked, we devised a state-transition diagram and implemented this in the prototype program, which is shown in Figure 8.

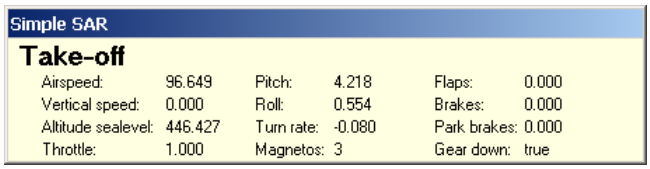

Figure 8: Screen shot of the prototype situation recogniser

In almost all on-line test cases, our prototype program was able to recognise the correct situation in real-time. However, in some cases the recogniser was a little late in detecting that the pilot was initiating landing procedures.

Expanding the prototype

Our next step is to expand the prototype situation recogniser program to accommodate a military aircraft such as the F16. Not only will this provide us with a more challenging and interesting domain with other situations, we also expect that the usage of an intelligent interface, which is our end goal, will have much more added value in a military aircraft than in a civilian airplane.

Rules and procedures about flying an F16 are well documented in two official F16 manuals available on the Internet [USAF 1996], [USAF 1995] and in the user manuals of the commercial flight simulator Falcon 4 [Microprose 1998], [Falcon unified team 2001]. These documents describe many situations that can occur during a military mission, as well as the actions that should be taken by the pilot in those cases. In order to have a more generic recogniser that can be used with multiple airplanes, we chose to encode the F16 rules and procedures in XML. The following situations have been described in our XML knowledge base [Mouthaan 2002]: start-up, taxiing, taking off, aborted takeoff, normal flight, dogfight, visual attack, non-visual attack, guided attack, harm attack, taking evasive action, deep stall, air refuelling, normal landing, flame-out landing, aborted landing, and shutting down. Since we now have to recognise a larger number of situations compared to the Cessna, we decided to use a slightly different approach. For every situation we designed a set of rules that produce a probability that that particular situation is occurring. The probability is calculated based on the state of the aircraft (FlightGear variables) or the recent events (pilot or environment). An event can have three sources:

Pilot: Pilot events are actions taken by the pilot, for example pushing a button or adjusting the throttle.

Aircraft: Aircraft events are changes in the aircraft's state, for example a change in altitude or speed.

Environment: An event from the environment can be a missile that is launched at the aircraft by an enemy SAM site.

Besides these three sources there is another source of information that can be used to determine the current situation, which is the flight plan. The flight plan contains information about the steer points the pilot should reach during the flight, but it also contains information about possible situations that will occur at those steer points (e.g. attack ground target). If the flight plan is entered in our system before the actual flight the system should be able to more accurately predict the current situation.

The rules

The rules are grouped according to the situation they relate to. Every rule has a value that indicates the probability that the rule accurately identifies the situation. When data (FlightGear variables) is passed to the knowledge base some rules will fire and some will not. A probability calculator will combine all the probabilities that are the result of the rules that fire and calculate a new probability for each situation. The probabilities that are stored in the knowledge base are fuzzy values from a fuzzy set. Once the probability calculators have produced a probability for every possible situation, an overall controller will evaluate all probabilities and determine if it can decide with enough certainty that one of the situations is taking place.

For every situation there are several types of rules:

Action rules: an action rule is a rule that states that a pilot has to or might perform a certain action during this situation.

Visual check rules: a visual check rule states that the pilot should check a certain instrument during the situation.

Conditional rules: the conditional rules can be used to determine if a situation has been started or if a situation has been finished.

Additional rules: rules that do not fit in any of the categories above.

Below we show an example of the XML code describing a dogfight situation:

```
<situation name="Dogfight" timewindow="30">
<actions>
    <phase name="ingress">
        <action name="fcr" priority="0/1" probability="vsp">&ACM;</action>
    </phase>
    <phase name="engage">
        <action name="master arm" priority="1"
        probability="BP"&MASTER_ARM;</action>
……...
</actions>
<visualChecks>
    <instrument name="HUD"/>
    <instrument name="radio/>
….
</visualChecks>
<constraints startProbability="SP" end Probability="BP">
    <constraint name="IFF" start="&OFF;" />
    <constraint name="RWR" start="&ON;" />
……
</constraints>
</situations>
```

CONCLUSIONS AND FUTURE WORK

We have presented some results of work in progress on an automatic situation recogniser in a flight simulator. We experimented with PCA analysis and neural networks to automatically recognise 7 states. The results were fairly good, but because of flexibility we decided to implement the situation recogniser as a knowledge-based production system. We devised a prototype situation recogniser that can detect the most common situations when flying a Cessna airplane. The prototype system also performed very well, except in some cases it was slow in detecting landing events. We have also shown our ideas about extending the existing recogniser to detect more complex situations (flying an F16) and adding probability values to the reasoning process.

The situation recogniser is part of a context-aware system that will be used in future research on intelligent interfaces in the cockpit. After our implementation of the F16 knowledge base and improved reasoning system, we plan to add a pilot-state recogniser module that should be able to assess the pilot's activities and workload.

Since our experiment platform, the FlightGear simulator, does not support multiple aircrafts yet, we are currently working on a multiplayer extension for FlightGear. Once the multiplayer extension, knowledge base, and pilot state recogniser are finished we plan to start experimenting with different intelligent interface strategies.

REFERENCES

Capkova, I., Juza, M. and Zimmerman, K. (2002) "*Explorative data analysis of flight behaviour*". Technical Report, Data and Knowledge Systems group, Delft University of Technology.

Falcon Unified Team (2001) "*Falcon 4 Superpak 3 User Manual"*, Infogames Inc.

Micropose (1998) "*Falcon 4.0 user manual"*, Infogames Inc.

Mouthaan, Q.M. (2002) "*Flying an F16: A knowledge base describing the situations an F16 pilot might encounter*". Technical Report DKS-02-03 / ACE 01, Data and Knowledge Systems group, Delft University of Technology.

Mulgund, S.S. and Zacharias, G.L. (1996) "A situation-driven adaptive pilot/vehicle interface", in *Proceedings of the Human Interaction with Complex Systems Symposium*, Dayton, OH, August 1996.

Perry, A.R. and Olson, C. (2001) "*The FlightGear flight simulator: history, status and future*", LinuxTag July 2001, Stuttgart, Germany.

USAF (1996) *Multi-Command Handbook 11-F16 (F16-combat aircraft fundamentals)*, Volume 5, May 10, 1996.

USAF (1995) *Multi-Command Instruction 11-F16 (Pilot operational procedures)*, PACAF Volume 3, April 12, 1995.