A RULE-BASED AND A PROBABILISTIC SYSTEM FOR SITUATION RECOGNITION IN A FLIGHT SIMULATOR

Patrick A.M. Ehlert, Quint M. Mouthaan *and* ***Leon J.M. Rothkrantz***
Data and Knowledge Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, the Netherlands

E-mail: {P.A.M.Ehlert, L.J.M.Rothkrantz}@ewi.tudelft.nl

KEYWORDS

situation recognition, artificial pilot, flight simulator, knowledge based systems.

ABSTRACT

In this paper we describe two situation recognition systems that have been developed for a flight simulator environment. The first system uses heuristic rules based on a state-transition diagram to determine the current stage of a flight. The second system does the same by calculating the probabilities of both the start and end of possible situations and determining the most probable situation. The idea is that the best situation recognizer system will be used as part of a more elaborate situation-aware system. This situation-aware system can be seen as a first step to an intelligent pilot bot.

1. INTRODUCTION

The Intelligent Cockpit Environment (ICE) project is a project of the Knowledge Based Systems group of Delft University of Technology. Originally, the main purpose of this project was to investigate techniques that can be used to create a situation-aware crew assistance system [Ehlert and Rothkrantz 2003][1]. Basically, a crew assistance system functions as an electronic co-pilot looking over the shoulder of the crew of an aircraft. This system tries to support the crew by providing useful information or taking over (some of) the crew's tasks if necessary. The idea is that this way the situation awareness of the crew will be improved and their workload reduced, leading to better and safer performance [Endsley 1999]. For this purpose we are investigating methods to create a situation-awareness module. The function of such a module is to create a "mental computerized picture" of the current situation. This mental picture includes aircraft status, flight progress, and crew performance among others. The situation-awareness module is used by the assistance system to make decisions when and how to support the crew.

Although, we are still investigating this application, our attention has also been drawn to artificial pilots that can be used for simulations. The idea is that the larger part of the situation-awareness module can just as well be used as the basis for decision-making of a simulated artificial pilot.

Our first step towards a situation-awareness system was to investigate the data that is available from a flight simulator [Ehlert, Mouthaan and Rothkrantz 2002]. Then we designed and tested some approaches to perform automatic recognition of situations based on this data. The goal of our situation recognition subsystem is to determine in real-time the status of the aircraft and the corresponding phase of the flight. In this paper we will describe two systems that we have created for this purpose. The first system uses heuristic rules embedded in a rule-based system. The second system uses probabilities to determine the most likely situation. Before we present both systems we will first discuss the related literature on artificial pilots.

2. RELATED WORK

We have found two different projects in the literature that deal with the construction of an artificial pilot, also called flight bot. The first one is TacAir-Soar. TacAir-Soar is an intelligent rule-based system that generates believable human-like pilot behaviour for fixed-wing aircraft in large-scale distributed military simulations [Jones et al 1999]. Each instance of TacAir-Soar is responsible for controlling one aircraft and consists of a Soar architecture [Laird, Newel and Rosenbloom 1987] linked to the ModSAF simulator. The interface between the Soar architecture and the simulator regulates the information that each aircraft receives from its own "sensors", such as aircraft status, radar, radio messages, etc. The advantage of using Soar is that the reasoning and decision-making of the system is similar to the way humans are generally believed to reason.

The second project dealing with the construction of flight bots are the intelligent air-to-air combat agents developed by the Linköping University in collaboration with Saab Military Aircraft AB in Sweden [Coradeschi, Karlsson and Törne 1996]. The system is designed specifically for air-to-air combat experts and allows them to specify the behaviour

[1] More information on the ICE project can also be found via http://www.kbs.twi.tudelft.nl/Research/Projects/ICE/

and decision-making of the intelligent pilot agents without the help of a system expert. The agents are modelled by decision trees. These trees contain production rules that describe the agent's dynamic task priorities. During one decision cycle, several branches of the tree can be processed in parallel after which all selected actions are evaluated for priority and compatibility. Due to the dynamic task priorities, sequential tasks that are spread over multiple decision cycles can be interrupted if the need arises.

Both TacAir-Soar and Linköping University's agents focus primarily on decision-making and both try to simulate realistic pilot flight behaviour. They do not specify how to deal with situation recognition or achieve situation awareness. Although achieving good situation awareness is not necessary to simulate the behaviour of (a large number of) artificial pilots, we feel that more realistic flight bots cannot do without. The better the understanding of the available data, the better a flight bot can deal with the current situation. By evaluating the current situation in real-time the flight bot can show much more flexible behaviour and come up with problem-solving strategies, resembling human reasoning.

After an extensive search, we have found one application in another domain that uses an approach similar to ours. [Nigro et al. 2002] describes two systems called Intelligent Driving Recognition with Expert System (IDRES) and Driving Situation ReCognition (DSRC). The goal of both systems is to provide support for a driving assistance system that is to be used in future cars. The DSRC system is able to recognize certain states of a manoeuvre performed by a car in a simulator. At a higher level, the second system called IDRES recognizes transitions between manoeuvres. Both systems are rule-based. Uncertainty of data is handled using fuzzy sets and beliefs on hypotheses.

3. THE GENERAL DESIGN

The ultimate goal of the flight bot in the ICE project is to create an intelligent system that has the knowledge, understanding and skill to fly an airplane, in the same way a human pilot does. We have devised a general architecture of this flight bot, which is shown in Figure 1. The bot uses decision cycles to read data from the simulator, create a representation of the current situation and decide which action to take. The function of the situation awareness module is to read all data coming from the simulated aircraft. This data is integrated with previous recorded information in order to create a representation of what is going on. Then, this world representation is used by the decision module to decide which action to take. The decision module can make use of several planners that are able to make predictions about future situations, for example the expected position of other aircraft. After the decision module has chosen an action to perform, this action is sent to the aircraft control manager. The aircraft control manager functions as an interface between the bot and the simulator. Ultimately, we want to be able to set certain properties of the flight bot, for example setting the level of pilot expertise so we can simulate different types of

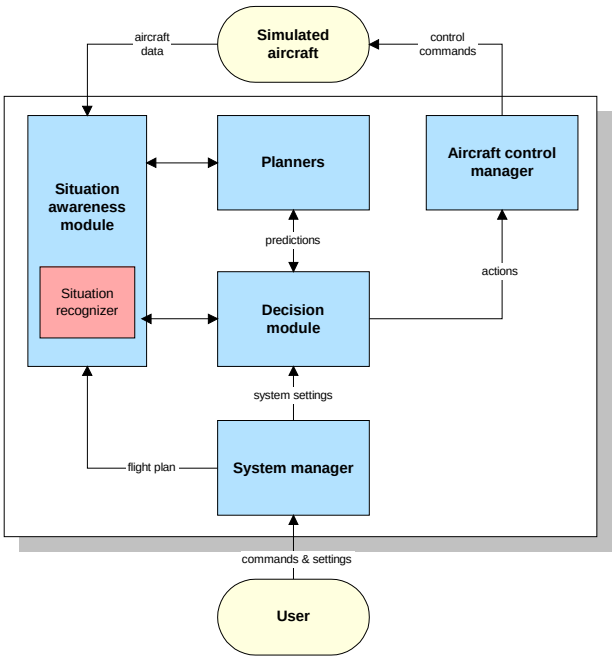

Figure 1: General architecture of the flight bot

pilots. This is the function of the system manager, which forms the interface between the user and the flight bot.

In the next sections of this paper we will discuss two situation recognition systems that we have devised as part of the situation awareness module: a rule-based system and a probabilistic system.

4. THE RULE-BASED APPROACH

One of our first attempts to implement the situation recognition subsystem was to use a rule-based approach. Rule-based systems, also known as production systems, allow simple, understandable, and transparent reasoning using IF-THEN rules. This makes rule-based system suitable for rapid prototyping, which is probably also the reason that they are one of the most popular methods of reasoning in artificial intelligence.

4.1 Design

The first step in the design of the rule-based situation recognition system was to gather knowledge on flying. Since there are many different types of aircraft we decided to restrict ourselves and start out by looking only at a simple and standard passenger aircraft: the Cessna 172C Skyhawk. Different types of situations during a flight were identified and for every situation the actions the pilot is expected to perform and typical situation-related variables were defined. We made rules for the following situations; pre-start, start-up, taxiing, hold-short, take-off, aborted take-off, set course, cruise, start-landing, aborted landing, final approach, touchdown and shutdown. All situations can be recognized based on a number of parameters such as airspeed, vertical speed, throttle, brakes status, gear status, etc. For each state

we tried to use multiple variables 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. While normally this is not necessary for an artificial pilot, it allows us to identify possible malfunctions, which we plan to add later to our situation awareness module.

To reduce the amount of rules that have to be checked every decision cycle, we devised a state-transition diagram, part of which is shown in Figure 2.

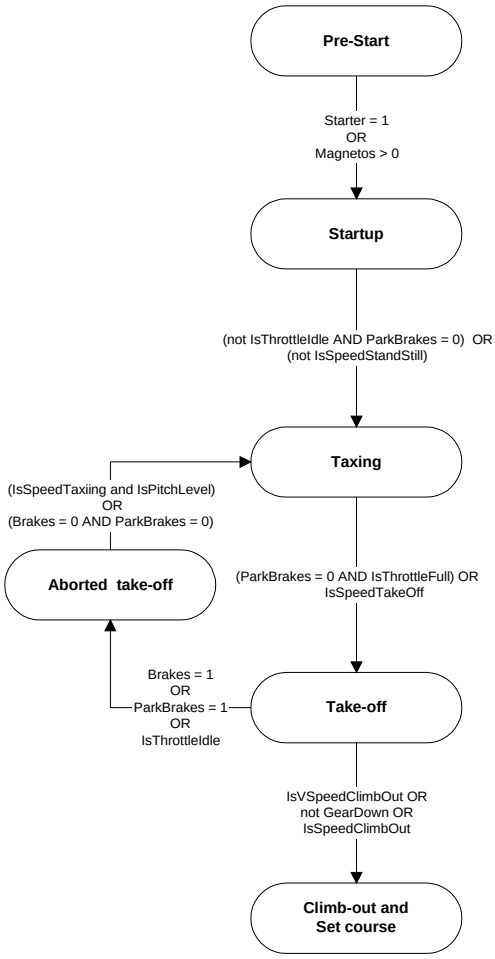

Figure 2: Partial state-transition diagram containing several situations of a flight with the Cessna airplane

The system is initialised in the Pre-Start state. Normally, only the rules belonging to this state are checked, until one of the rules changes the state to Startup. Then only the rules belonging to the Startup state are checked, etc. To make the system more robust, states are changed not only when evidence is found for a state transition (a new situation has arisen), but also when there is evidence that the current state cannot be the correct one. In this case we have to make a decision which connecting state is most likely.

4.2 Implementation

The rule-based system was implemented with Borland Delphi 5 and the simulator we used to test the system was Flightgear, version 0.7.10 [Perry and Olsen 2001]. Newer versions of the open-source Flightgear simulator are available, but proved to be less stable.

The rule-based system receives information from the simulator about the state of the airplane (e.g. airspeed, altitude, pitch), the actions of the pilot (e.g. setting flaps, pushing the throttle), and the environment (e.g. wind), all via a Telnet connection. The rules and state-transition diagram were hard-coded into our program using IF-THEN statements. Using hard-coded rules has the advantage that reasoning can be done very fast. There is less overhead compared to using a third-party rule-based system such as CLIPS or JESS. However, it can be difficult to alter rules or add new rules, especially when the rule-base is large.

Below we show (part of) an example rule corresponding to the Take-off state in Figure 2:

```
procedure StateTakeOff;
begin
   if IsVSpeedClimbOut(VertSpeed) or
      (not GearDown) or
      IsSpeedClimbOut(AirSpeed)
   then
      State := sSetCourse;

   if IsThrottleIdle(Throttle) or
      (Brakes = 1) or
      (ParkBrakes = 1)
   then
      State := sAbortTakeOff;
end;
```

Due to some difficulties with the Telnet connection between our program and Flightgear we were only able to retrieve data from the simulator about once every 500 ms. This is a fairly large timeframe and it is possible that the system misses certain events that have a shorter duration. This is another reason that we check multiple variables (besides identifying possible malfunctions which we mentioned earlier).

5. THE PROBABILISTIC APPROACH

One of the disadvantages of using a rule-based system is that IF-THEN rules are always deterministic. Either the IF-condition of the rule is fulfilled or it is not. A certain event or variable value may be an indication for more than one situation. For example, a pilot can reduce the throttle if he wants to land, but also simply to reduce speed and save fuel. We tried to solve this problem by introducing probabilities to determine the likelihood of the start and end of each possible situation.

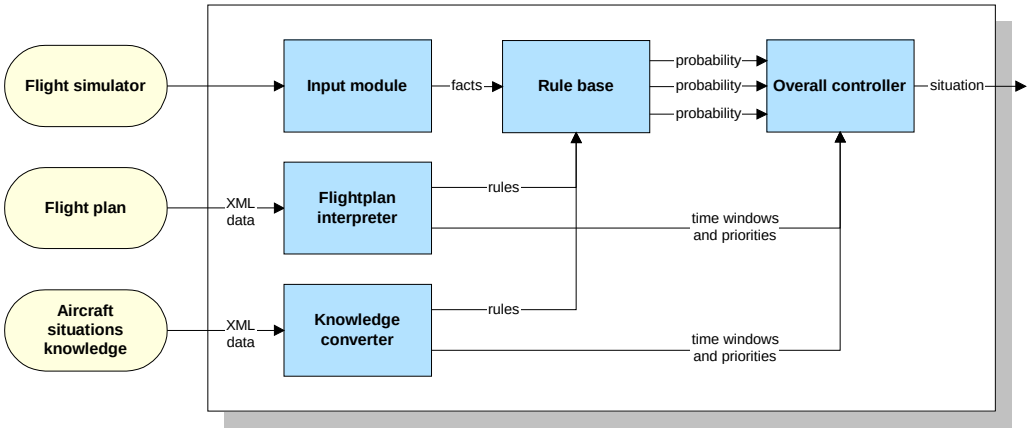

Figure 3: Architecture of the probabilistic situation recognizer

5.1 Design

Our probabilistic approach extends the rule-based approach described in the previous section in a sense that now the rules are not used to detect a situation deterministically, but to generate probabilities about situation starts and endings. Since this requires us to check multiple situations at the same time, the state-transition diagram was abandoned. However, to reduce the number of rules that need to be checked, we added preconditions that need to be fulfilled for each situation. For example, for the taxiing situation to occur, the landing gear has to be down.
Another extension in our probabilistic system is that we have added the possibility to load different rules for different aircraft. The architecture of the probabilistic situation recognition system is shown in Figure 3.

The **knowledge converter** converts all the situations knowledge for a particular aircraft stored in an XML file to IF-THEN rules. These rules are loaded into the rule base before the recognition system is started.

The **flight plan interpreter** converts the information in the flight plan to a number of rules that are put in the rule base. These rules can help situation recognition by predicting which situations will occur in the near future. Just as the aircraft situations knowledge, the flight plan is loaded before a flight.

During a flight, the **input module** receives aircraft data from the flight simulator and converts this data to facts that are forwarded to the rule base.

The **rule base** contains all the rules and facts that have been generated by the flightplan interpreter and knowledge converter. When data (facts) from the flight simulator are added to the rule base, some of the rules will fire and generate probabilities concerning the start or end of a situation. These probabilities are then passed to the overall controller.

The **overall controller** receives event data and situation probabilities from the rule base, combines these probabilities, and calculates for every situation the probability that it has started or the probability that it has ended. It then draws a conclusion about the situation that is most likely to be the current one. Calculating probabilities is done using a probabilistic network.

5.1.1 The start probability calculator

Figure 4 shows the probabilistic network that is used to calculate the probability that a situation has started.

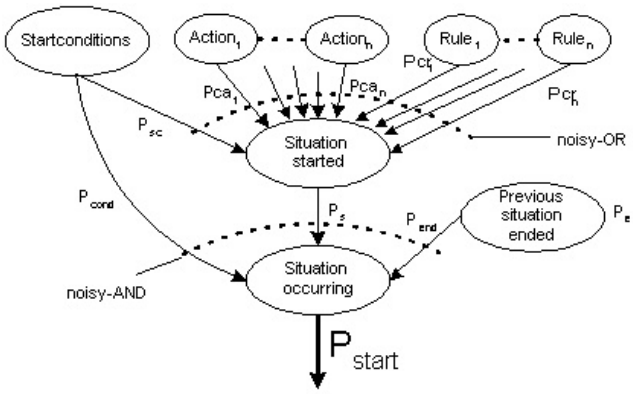

Figure 4: Probabilistic network that calculates the probability that a situation has started

The **start conditions** for a situation are conditions that must be satisfied before a situation can possibly have started, otherwise the probability that the situation is started will be zero. When the start conditions are satisfied, the probability of the conditions that is specified in the aircraft situations knowledge will be the output of this node.

The **action probabilities** are passed to the probability network by action rules that are activated when the pilot performs a particular situation-related action. Action rules are rules that check if an action has been performed that belongs to a situation. An action rule can only fire if the start conditions of a situation have been met. All action

probabilities contribute to the probability that the situation is occurring (has been started).

The **additional rules** are rules that fire when the state of the aircraft changes or when a specific event occurs. They also include rules similar to the consistency checks used in the rule-based approach that check if the current state is still the correct one. When an additional rule fires, it generates a probability that the situation has started or ended.

The **probability calculator** (situation-started node in the figure) combines the probabilities of the nodes that have been described above using the noisy-OR model.

The **previous situation** influences the start probability of a situation. The idea is that the probability that a situation is occurring increases when the probability increases that one of the previous situations that can lead to this situation has ended.

Based on this network the probability that a situation has started and is occurring can be calculated with the following formula:

$$P_{start} = P_{cond} * P_{end} * P_s$$

$$P_{cond} = \begin{cases} 0 & if\ P_{sc} = 0 \\ 1 & if\ P_{sc} > 0 \end{cases}$$

$$P_s = 1 - ((1 - P_{sc}) * \prod_{i=1}^{n} (1 - Pca_i) * \prod_{j=1}^{n} (1 - Pcr_j))$$

In this formula, P_{sc} is the probability of the start conditions, P_{end} is the probability that one of the previous situations has ended, Pca_i is the probability of the i-th action that should be performed during the situation and Pcr_j is the probability of the j-th event or state change that can occur during the situation.

5.1.2 The end probability calculator

In Figure 5 the probabilistic network is shown that calculates the probability that a situation has ended. In this network we see a lot of the same nodes as in the network for the start of the situation. The nodes that are different are discussed below.

The **time window** for a situation is the maximum duration of that situation. If the start of a situation has been detected the probability that it has ended should grow after a certain time.

The **situation-started** node produces a 1 if the situation has started and a 0 if the situation has not yet started. This node is necessary because we only want to calculate the probability that the situation has ended, after a (probable) start of that situation.

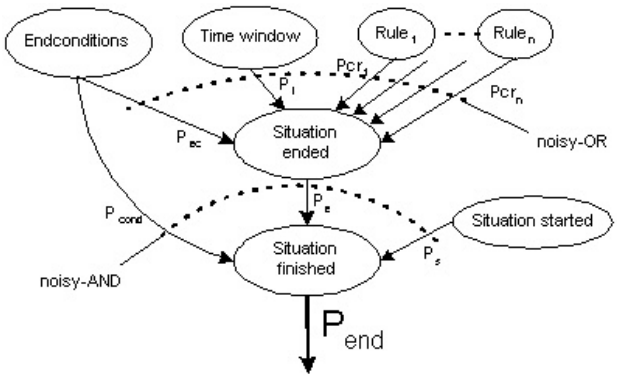

Figure 5: Probabilistic network that calculates the probability that a situation has ended

The probability that the situation is ended can be calculated with the following formula:

$$P_{end} = 1 - ((1 - P_{sc}) * (1 - P_t) * \prod_{i=1}^{n} (1 - Pcr_i))$$

More details about the design of our probabilistic system can be found in [Mouthaan 2003].

5.2 Implementation

The probabilistic system was implemented in Java. Unlike the rule-based system, the rules were not hard-coded since we wanted to be able to load different rules for different aircraft. Therefore, we chose to use the JESS rule-based system [Friedman-Hill 1997]. The system was tested using Microsoft's Flight Simulator 2002, which was found to be more realistic and stable than the Flightgear simulator that we used earlier with our rule-based recognition system. Flight Simulator 2002 allows retrieving data from the simulator by an external program via a shared memory space. The interface between the simulator and the system was implemented with C++. We have devised a situations XML file for the military F-16 aircraft and the civilian Cessna C172 plane. Using these aircraft, we have performed several experiments to test the system. The flightplan interpreter that was described in the design of the probabilistic system was not implemented yet.

6. EVALUATION

We have evaluated a prototype version of both situation recognizer systems by performing several flights and logging the results. The time and name of all detected situation changes were logged, as well as the time a new situation started according to the pilot. This allowed us to check if the systems recognized a situation correctly and in time. However, one must note that the time a situation starts is often a bit vague and subjective. For example, the change between the situations "climb out/set course" and "normal flight" is difficult to pinpoint precisely. For this reason we have rounded off all recorded times to whole seconds.

The test results of both the rule-based system and probabilistic system were fairly good. On average, the rule-based recognizer detects situations one or two seconds after they occur. The rule-based system sometimes has some difficulties detecting the start of the landing situation. The probabilistic recognizer performs similarly. It has less difficulty with the landing situation but for some unknown reason it sometimes detects the normal flight situation several seconds before it actually occurs. In one of our experiments we found that the probabilistic system is even able to correct a mistake immediately in the next reasoning cycle. The mistake occurred during landing, when the recognizer inadvertently thought the landing was being aborted. We suspect the mistake was made due to an error in the rule base that resulted in keeping a fact in the rule base for too long.

In Table 1 and 2 we have presented some results of both systems on one of our simpler flights, which was to fly a standard circuit with the Cessna 172C. Flying a circuit means that the pilot has to take-off, circle around to the beginning of the runway and land again (see also Figure 6).

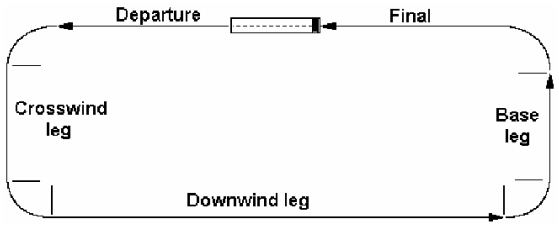

Figure 6: A standard circuit

The first column in both tables contains the names of the situations that occurred or were detected by the system. The second column contains the times at which the pilot considered the situations to be started. The third column contains the times at which the situations were detected by the recognition system. Times are given in seconds from the moment the program was started. Note that comparing the two presented tables is not entirely fair since the flights were flown by different pilots and on different simulators. However, at the moment we have no better way of comparing the two systems.

From the tables it can be seen that both systems did not make any serious errors and were able to recognize most situations in a matter of seconds. The probabilistic situation recognizer performed slightly better (less time wrong) than the rule-based recognizer. We have calculated the error rate of the flights by dividing the amount of time that a recognizer was incorrect by the total time of the flight. However, in this case the pilot of Table 1 took longer to complete the circuit than the pilot in Table 2, so therefore the error rate of Table 1 is lower even though the recognizer was incorrect for a slightly longer amount of time.

Other experiments that were performed showed similar results. On average the error rate over the performed experiments for the probabilistic recognizer (4 flights) was 0.08, with 0.05 being the lowest recorded error rate and 0.11 being the highest. The rule-based recognizer reached an average of 0.09 over 5 flights, with 0.03 as best and 0.15 as worst. As mentioned earlier, the comparison between the two systems is inconclusive, but the results seem to indicate that the probabilistic approach performs slightly better than the rule-based approach.

7. CONCLUSIONS AND FUTURE WORK

We have created two systems that can recognize the current situation during a flight with a simulated aircraft. The first system uses a combination of a rule-based approach and state-transition diagram. The system was able to detect situations flying the Cessna aircraft. The second system is based on a probabilistic model. This system can load a model for a specific type of aircraft before the flight. Currently we have a model for both the F-16 and the Cessna aircraft.

Investigating a number of test scenarios, both systems seem to work fairly well. They make few mistakes and are even able to correct them immediately. Furthermore they are able to come to a conclusion about the current situation in real-time. We have tried to compare the results of both systems, but this comparison is complicated due to the vagueness of the exact start of a situation, and the different pilots and simulators we used. We are currently busy redesigning the rule-based system to work with the same simulator used in our experiments with the probabilistic recognizer, so we can make a more accurate comparison between the two systems.

Table 1: Results of a standard circuit flight with a Cessna using the rule-based situation recognizer

Situation	Time started (s)	Time detected (s)
Start-up	0	0
Taxiing to runway	16	17
Taking off	27	29
Normal flight	73	72
Landing	187	179
Taxiing from runway	272	275
Shutdown	302	302
Error: 15 seconds (5,0%)		

Table 2: Results of a standard circuit flight with a Cessna using the probabilistic situation recognizer

Situation	Time started (s)	Time detected (s)
Start-up	0	0
Taxiing to runway	7	10
Taking off	22	27
Normal flight	61	59
Landing	119	121
Taxiing from runway	220	221
Shutdown	251	251
Error: 13 seconds (5,2%)		

Although the test results of both systems are fairly good, we are not yet satisfied. It often takes a few seconds to accurately detect a situation. This is no problem for a Cessna, but in an F-16 covering more than 500 meters per second, this can be a problem. Therefore, we would like to detect situation changes almost immediately. Our future work will consist of improving and fine-tuning the rules used by both systems to detect situation changes faster and increase reliability. One way to do this is to use the flight plan to help detect the current situation and predict future situations. However, it is possible that pilots deviate from the flight plan, so this should be included in the recognition process. In addition we would like to expand the systems to include more detailed, synchronous situations and specific events such as malfunctions. Together with our other efforts currently underway to implement a decision module and planner modules, we want to use this improved situation awareness module to create a human-like intelligent flight bot.

REFERENCES

Coradeschi, S., Karlsson, L. and Törne, A. (1996) "Intelligent agents for aircraft combat simulation", in *Proceedings of the 6th Computer Generated Forces and Behavioral Representation Conference*, pp. 23-25 July 1996, Orlando, Florida, US

Endsley, M. R. (1999) "Situation awareness in aviation systems", in *Human factors in aviation systems*, Garland,D.J., Wise, J.A. and Hopkin, V.D. (Eds.), pp. 257-276, Lawrence Erlbaum.

Ehlert, P.A.M. and Rothkrantz, L.J.M. (2003) "*The Intelligent Cockpit Environment Project*", Research Report DKS03-04/ICE 04, Knowledge Based Systems group, Delft University of Technology, The Netherlands.

Ehlert, P.A.M, Mouthaan, Q.M. and Rothkrantz, L.J.M. (2002) "Recognising situations in a flight simulator environment", in *Proceedings of 3rd Int. Conference on Intelligent Games and Simulation* (GAME- ON 2002), London, Great Britain, pp. 165-169.

Friedman-Hill, E.J. (1997) "*JESS, the rule engine for the Java platform*", JESS manual for version 6.1, Sandia National Laboratories, http://herzberg.ca.sandia.gov/jess/docs/61 (link checked November 6th, 2003)

Jones, R.M., Laird, J.E., Nielsen, P.E., Coulter, K.J., Kenny, P. and Koss, F.V. (1999) "Automated intelligent pilots for combat flight simulation", in *AI Magazine, Vol. 20*, No.1, pp 27-41.

Laird, J.E., Newell, A. and Rosenbloom, P.S. (1987) "Soar: an architecture for general intelligence", in *Artificial Intelligence, Vol. 33*, No.1, pp. 1-64

Nigro, J.M., Loriette-Rougegrez, S. and Rombaut, M. (2002) "Driving situation recognition with uncertainty management and rule-based systems", in *Engineering Applications of Artificial Intelligence, Vol. 15*, pp 217-228, Elsevier Science Ltd.

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

Mouthaan, Q.M. (2003) "*Towards an intelligent cockpit environment: a probabilistic approach to situation recognition in an F-16*", MSc. thesis, Knowledge Based Systems group, Delft University of Technology, The Netherlands

AUTHOR BIOGRAPHY

PATRICK EHLERT has obtained his Master's degree in Computer Science at the Delft University of Technology. There he now works as a PhD student on the ICE project.

QUINT MOUTHAAN has obtained his Master's degree in Computer Science at the Delft University of Technology. He recently started working as an engineer at Force Vision, a company developing mission-critical systems for the navy.

LEON ROTHKRANTZ has a degree in psychology and mathematics and is working as a lecturer at the Delft University of Technology.