Creating a dogfight agent

David Solinger, Patrick Ehlert, and Leon Rothkrantz
Department of Media and Knowledge Engineering
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
E-mail: {D.Solinger, P.A.M.Ehlert, L.J.M.Rothkrantz} @ ewi.tudelft.nl

Abstract

This paper describes the design and the implementation of a prototype for a fully autonomous agent, which can fly an airplane during a dogfight. This project focuses on combining multiple elements of flight automation, such as situation recognition, decision-making and maneuver execution in one fully autonomous agent. A flexible and modular overall architecture is presented in which, for each element of the flight automation process, different methods can be implemented and replaced easily. The methods that are used for the prototype implementation are presented with a detailed design and a description of the implementation. The prototype is successfully implemented and can really act as a competitive player during a one-to-one dogfight in Microsoft Combat Flight Simulator. An important conclusion is that the performance of the whole agent is dependent on the performance of all the different elements. This conclusion underlines the relevance of this project, because the basis of the project was to split the concept of flight automation into multiple elements, so it is possible to focus on the individual elements.

1. Introduction

Since the first airplane was build, more than 100 years ago, the capabilities of aircraft have been improved enormously. Together with the increase in the amount of aircraft in the air this has caused a huge increase in the information load for a pilot. In situations like this, humans normally start to look for ways in which a computer can help them. So projects have been started to investigate and design human-support systems for use in a cockpit. The intelligent cockpit environment (ICE) project is a project of the Knowledge Based Systems group of the Delft University of Technology. The goal of the ICE project is to design, test and evaluate computational techniques that can be used in the development of intelligent situation-aware crew assistance systems [Ehlert, 2003]. This paper described a part of the ICE project in which a fully autonomous agent is created.

Within the ICE project different areas of flight automation have been addressed, like: situation recognition, decision-making, flight planning and flight control. This resulted in a number of ideas and methods for modern flight automation. Most of them have successfully been tested, but not in combination with other ideas. The ideas are not combined or the influence of one task on another is not studied. For example, a good situation recognition method can maybe improve the decision-making. In this project multiple ideas and methods of the ICE project are combined in one program. All necessary tasks are added, so that it can act completely by itself, without any human interventions. This will result in an architecture in which new methods for flight automation can be designed, developed and tested. Furthermore this project focuses on the decision-making process and the execution process. These parts of flight automation are already addressed before in the ICE project, but the current techniques are not good and/or not detailed enough to be usable.

The goal of this project is to create an agent that can ‘play’ the computer game Microsoft Combat Flight Simulator (MSCFS). In this simulator game the agent has to be a competitive player in a one-to-one dogfight. The dogfight is one of the most difficult scenarios for flight automation, so that is why this scenario is chosen. The opponent of this agent can be either a human player or another computer program. The agent should combine methods discovered in the ICE project in a flexible design and a nice implementation so that the program can be used as a test environment for future research.

2. Related work

ICE project

The goal of the ICE project is to design, test, and evaluate computational techniques that can be used in the development of intelligent situation-aware crew assistance. Special issues addressed in the ICE project are: situation recognition, mission or flight plan monitoring, attack management and pilot workload monitoring. Until now, most work had been done in the field of situation recognition. Different techniques are studied and tested like Bayesian belief

networks [Mouthaan, 2003] or neural networks [Capkova, 2002]. However this work was not applied to dogfight situations. Another study within the ICE project [Andriambololona, 2003] is applied to dogfight situations. In this study about decision-making for a dogfight agent a decision-tree is modeled after decisions as they are made by human pilots. This report describes very well how a human pilot basically makes decisions in a dogfight situation, however the described model is not yet detailed enough for a fully autonomous agent. A last study within the ICE project that will be mentioned here is about the construction of an agent that can fly a human-defined flight plan [Tamerius, 2003]. The agent is able to execute a set of consecutive flight steps in a simulated flight environment.

Flight automation

Besides the ICE project there are many other projects that focus on flight automation in one way or the other [Laird, 1998], [Virtanen, 2001]. In most of the projects a simulated environment is used to test new ideas. Many of the interesting projects are about strategy (or decision-making). In many projects the dynamic, complex and real-time environment is seen as the biggest challenge of the project. The locations of the airplane and the opponent(s) change constantly, also when no explicit action is taken. Furthermore the environment is real-time which means that there will never be a lot of time for reasoning so that time-consuming algorithms are not usable. One of the interesting projects is about agents that can fly military missions in a military simulator [Laird, 1998]. The agents are created to create both enemies and friends for human pilots. The decisions in this project are mainly on a higher level than decisions in a dogfight situation. For example a rule in this system can be ‘if an enemy plane is in front of us, go into dogfight’. It is not clear how this dogfight will be executed. This is often the case in projects about decision-making in flight automation. In many projects there is no implementation created, so there is no need to specify more detailed and lower-level decisions.

AI in computer games

Another field of interest related to creating a dogfight agent for a simulated environment is the field of AI in computer games. In recent years the interest in AI in computer games grew for several reasons [Tozour, 2002]. One reason is that because of the increasing complexity of modern computer games, it becomes more difficult to create interesting digital opponents for human players. Another important reason for the increasing interest in AI in computer games comes from the academic world. It is far more easy, fast and cheap to test new ideas in a simulated environment, such as a computer game. An interesting project in this area is the development of the Quake III Arena Bot [Waveren, 2001]. In this project an artificial player is created for the computer game Quake that can serve as an interesting and challenging opponent for human players. The AI of the bot is based on proven and good-to-understand AI techniques like a finite state machine and a rule-based system. Despite the success of quite basic AI techniques in computer games, there is also research on the use of more sophisticated AI techniques in computer games. In [Spronk, 2003] neural networks are used in combination with genetic selection. However the success of projects like these depends heavily on the simplicity of the computer game. Until now these more sophisticated AI techniques are tried out in more complex computer games only for subtasks.

3. Design

Model

In a one-to-one dogfight situation both pilots have the same goal: eliminate the enemy and stay unharmed yourself. In this project the actual shooting is left out. So the goal of the agent will be to get and stay in the best shooting position, which is close and straight behind the opponent. To reach this goal the agent can execute different flight maneuvers by changing the elevator or aileron controls of the airplane. The situation of both airplanes and the relation between them changes continuously. The chosen model for the agent is based on the model of a reflex agent. The agent percepts, evaluates and acts in very small time steps, so its behavior comes close to a continuous behavior. Figure 1 shows how the agent and the environment are related in the agent model.

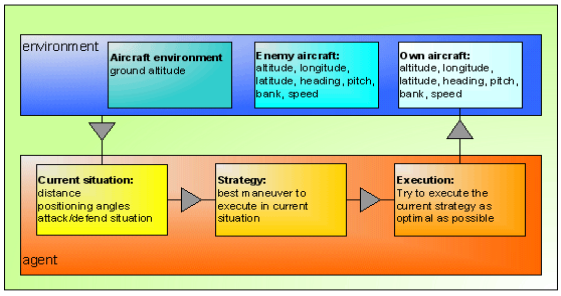

Figure 1: Model of the dogfight agent

System Architecture

The simulated environment will consist of two computers, participating in a MSCFS multiplayer game over a TCP/IP network. On both computers the agent software must run, so communication is possible between the two agents. This setup is shown in Figure 2. In this setup two agents can play against each other, but it is also possible that a human player takes over control on one of the computers, so one agent is playing against one human. There is also some communication between the two bots. This is because it was not possible to retrieve data about the opponent directly from the MSCFS.

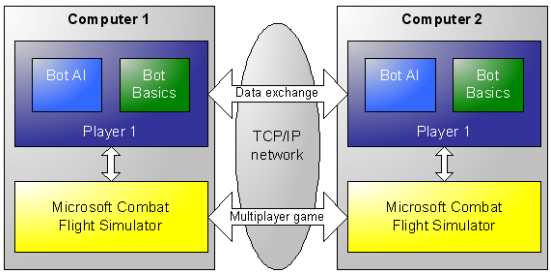

Figure 2: System architecture

Agent architecture

One single agent is designed in a number of objects and five layers. Figure 3 shows these layers and objects.

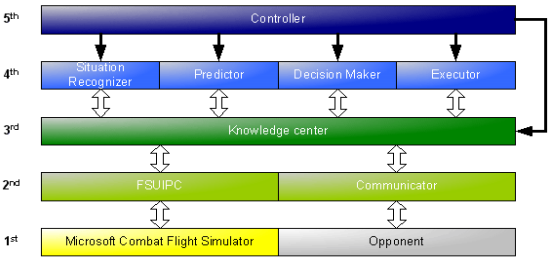

Figure 3: Agent architecture

The first layer contains the external objects: The MSCFS game and the opponent. These two objects do not form a part of one single agent itself, but they are essential for the proper working of the agent.

In the second layer there are two interfaces towards the two external objects. The FSUIPC object consists of a module that provides data exchange between the agent program and MSCFS. The Communicator is the object that communicates with the opponent via a TCP/IP network.

The whole third layer is formed by one object: The Knowledge center. This object keeps tracks of all data from both lower and higher layers. All necessary data from the external objects is collected via the FSUIPC and Communicator objects. This data is available for the objects in the fourth layer. All data produced by objects in the fourth layer is in the Knowledge center available for the external objects, via the interfaces, and also for the other objects in the fourth layer.

There are four objects in the fourth layer. The Situation Recognizer recognizes the current situation by looking at all the data collected in the Knowledge center about the current situation, for example the current locations of both airplanes. The recognized situation is stored in the Knowledge center. The Predictor must predict future values of certain variables based on their current and past values. Because of practical reasons the Predictor is not implemented or used further on in this project. The Decision-maker decides which maneuver will be executed in the next time step. The Executor calculates the right values for the airplanes input controls in MSCFS. These values are also stored in the Knowledge center. Via the FSUIPC object these values are put into the MSCFS, which makes the airplane execute the chosen maneuver.

The only object in the fifth, and highest, layer is the Controller. The only task of this object is to schedule all the tasks of the other objects each time step in the right order.

Situation recognition AI

Situation recognition is the first intelligent task of the agent. Based on data retrieved directly out of the environment and also on data that is calculated out of this easily, such as the range between the two airplanes or the angle between the directions of the two airplanes, the situation is recognized. In the dogfight agent situation recognition consists of two things: recognizing the maneuver the opponent is executing and recognizing the relation between the positions of the two airplanes. Maneuver recognition can be done in various ways, for example with a Bayesian belief network [Mouthaan, 2003] or a neural network [Capkova, 2002]. To be able to create a successful prototype, a rather simple AI method: a decision-tree is chosen for maneuver recognition. Altogether, nine maneuvers can be recognized with the decision-tree from which a part is shown in Figure 4.

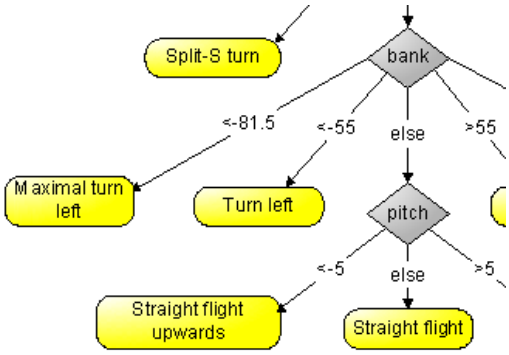

Figure 4: Part of maneuver recognition decision-tree

The relation between the positions of both airplanes is also recognized with a decision-tree. A part of this decision-tree is shown in Figure 5. There are 15 different position relations in this decision-tree, for example: ‘Long distance’, ‘Middle distance – attack’, or ‘Small distance – defend’.

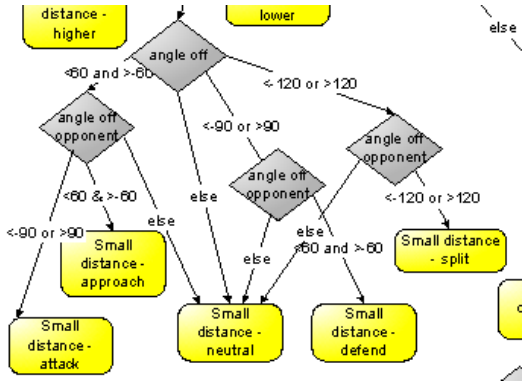

Figure 5: Part of position recognition decision-tree

Decision-making AI

Based on the current situation, the agent needs to decide the next maneuver to execute. The decision-making will be done by a set of logical rules. The big advantage of rule-based decision-making in a first prototype is that it is easy for humans to understand the rules. Even, when a computer algorithm creates the rules. This makes analyzing unexpected behavior much easier.

The most difficult task in creating a rule-based system is the acquisition of the rules. There are three ways of acquiring rules: the rules can be entered by domain experts [Andriambololona, 2003], they can be learned by observing experts [Lent, 1998] of the rules can be learned by experience. The last two methods are more complex to design and implement, especially the third one. Because of the goal to create a working prototype and because of the scope of this project the first method is chosen. This method is easier to implement, but it requires a certain amount of domain knowledge.

To make the task of entering rules a bit less complex, the decision-making system is split up into 15 smaller sets of decision rules, one for each position situation as it is recognized by the Situation Recognizer. In one specific situation only a subset of all available situation data items is relevant. Also only a subset of the maneuvers is useful. Besides that, the domain expert is working with one specific situation at a time, therefore the expert can fully focus on the specifics of that situation. Figure 6 shows that the rule-based system for the ‘Long distance’ situation works with six relevant input variables and can lead to only three of the nine possible maneuvers.

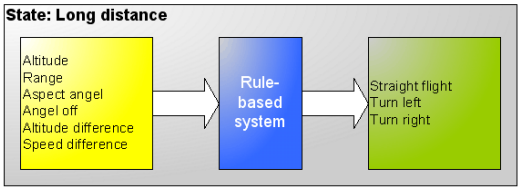

Figure 6: Rule-based system for Long distance

Execution AI

The decision-making system decides which maneuver will be executed. After this the selected maneuver must be executed by an execution system. The agent cannot order the aircraft directly to make, for example, a turn left. The agent can only use the elevator and aileron parameters to perform the desired maneuver. Most maneuvers differ from each other, not only in their result, but also in the approach of the pilot. Therefore the execution of each maneuver will be separately designed and implemented.

How will the agent know which values for the aileron and the elevator must be set to perform the right maneuver? In [Tamerius, 2003] the values are changed a little bit each time step and the results are monitored. The airplane will fly in the desired position, after enough time steps. This method is not fast enough for a dogfight situation. In [Liang, 2004] a neural controller is used, but the results show that it is still not very easy to create a highly reactive agent for a dogfight situation. In this project the execution is created by learning from experts. From flights by human experts a log is created that contains for each time step the values of the input control parameters and the corresponding airplane behavior. In these flights the human expert executed all the necessary maneuvers, but also some complete random flights.

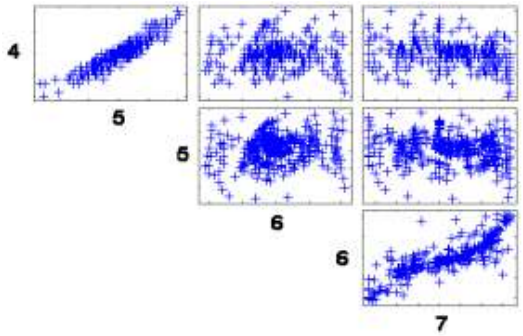

Figure 7: Relations between speed change (4), pitch (5), pitch change (6) and elevator (7) in a random flight

Figure 7 shows that even in a random flight there is a correlation between some parameters. For example the relation between the input control ‘elevator’ and the ‘pitch change’ is very useful.

In this project polynomial regression is used to find mathematical functions that express the relation between the desired output and the input control parameters. This method resulted in a good working and very fast maneuver execution.

The execution is optimized by making a difference between being in a desired maneuver position, and not being is a desired maneuver position. a situation in which the airplane already is executing a desired. When the airplane is not yet flying the desired

maneuver it must first get in the desired position. After, about two or three time steps, a little bit less than one second, the airplane will get in the desired position. When by that time, still the same maneuver is desired, the agent can fully focus on executing that maneuver as precise as possible. Different mathematical mappings are created for this situation, because now the airplane must try to continue this maneuver as precise as possible. In a dogfight, not only the tactics, or decision-making, makes the difference between the two opponents. The player that can execute the maneuvers best also has an advantage.

4. Implementation

The prototype is implemented in Microsoft Visual C++. The implementation follows the design closely. Each object in the agent architecture is implemented in a C++ class. The active objects are implemented in a thread.

The Knowledge center retrieves the data via the FSUIPC module out of MSCFS. This is not a very complex task and it is better to have very accurate data available for the other objects, so this thread runs every 100 milliseconds. The data exchange with the opponent in the Communicator also runs individually every 200 milliseconds.

The controller makes calls to all the objects in the Intelligence layer and the Knowledge center in the right order. This is done every 300 milliseconds. This interval is chosen after some experimenting. A much larger time interval makes the agent less reactive, but the time interval can also not be much shorter. The airplane simply needs some time to start reacting, before it is useful to monitor the situation and make a new decision again.

A more detailed description of the implementation is included in [Solinger, 2005]. That report also includes some source code examples.

5. Results

The implementation of the dogfight agent is used in a few test scenarios in a MSCFS multiplayer game. In the first test scenario the human opponent of the agent just flies straight ahead, in the second test scenario the opponent makes some easy turns in both directions. In both tests the agent could easily follow the opponent and get in the best position, close behind the opponent. Figure 8 shows two top views of consecutive time periods in the second test scenario. From these two test scenarios we can conclude that the agent is able to intercept the opponent. The agent must be able to go through the whole reasoning process and take each step in good order to be able to follow the opponent and get close behind him.

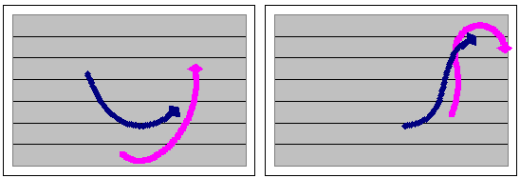

Figure 8: top view in test scenario 2

The third test scenario is a heavy dogfight. In this scenario the human opponent also tries to attack and he tries to use all kinds of very steep maneuvers to defeat the agent. Both players start by flying towards each other. After they are close enough, the agent and the human player both start turning around each other to get behind the other. The agent has a better overview, makes the right decisions and performs the maneuvers slightly better than the human player. Figure 9 shows that the agent comes in an attack position close behind its opponent. Thereafter the human player tries to escape by making very steep maneuvers. The agent is able to follow and stay in the ‘small distance attack’ situation for more than 60 seconds.

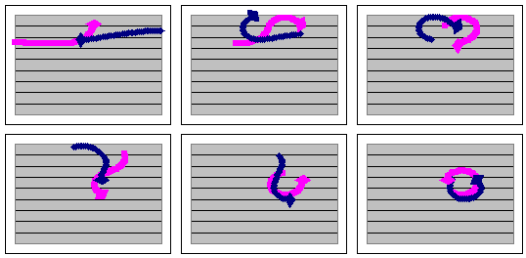

Figure 9: first 42 seconds of the dogfight

The human player manages to escape by executing a maneuver the agent does not understand: flying almost straight down. The agent does not make the best decisions in this situation and the human player is able to get in an attack position for a few seconds. The agent comes in a defend position, but it recognizes the situation and is able to regain control over the situation. The agent seems to be better in standard situations, because it gets rid of the human player in the same way as during the start of this dogfight. The agent performs its maneuvers just a bit better, and it does not make mistakes in the situation recognition and decision-making phases as long as the human player does not do something really unexpected.

Figure 10 shows the situations according to the agent during the dogfight. One can see that the agent was more often in an attack situation than in a defend situation. This figure also shows that the agent’s situation is relatively stable. It is important that the situation does not switch between different states all the time, because this could result in unstable and not effective behavior.

Figure 10: situations during the dogfight

6. Conclusions

The overall performance of the agent turned out to be satisfying. The prototype is fully functional and has become a 'competitive player' in a dogfight situation. It works real-time on an everyday pc. Especially in standard situations the agent's reactions and maneuvers were faster compared to its human opponent. On the other hand a human player can be more creative and execute for the agent yet unknown or at least unexpected maneuvers. Altogether the current state of the agent can be compared to the first check computer programs. The agent always reacts properly and fast in known situations, this can be a large advantage when a human opponent makes a small mistake or looses concentration for a few seconds. The only way for a human player to defeat the agent is to use its ability to be more creative than the agent.

The performance of the complete agent is dependent of the performance of each of the individual aspects of the flight automation process. This conclusion underlines the relevance of this project. By splitting the whole automation process into multiple elements it is possible to improve each of these individual elements. During the development of the prototype the performance of the bot increased significantly. Sometimes improvements in only one of these objects resulted in a big improvement in the performance of the bot. For example a small improvement in the 'turn' maneuvers once resulted in a competitive bot instead of a strange behaving airplane. It is also important to combine the elements in the right way to fully use the power of each of the individual elements.

This project resulted in a flexible architecture in which multiple objects can be altered, replaced or added without changing the other objects. This makes it very suitable for testing new ideas in future work.

7. Future work

The current prototype really works and the agent's results during the test scenarios are satisfying. However there still are plenty of ideas for future work. Three of them are mentioned.

Improve the decision-making

The first improvement for the agent is quit obvious. In test scenario 3 the agent got into problems, because it was unable to deal with an unknown situation. The easiest way to overcome this problem is to create more and better rules in the decision-making system for all kinds of situations.

Advanced situation-based decisions

The decision-making is already situation-based. This could be used even more. Different situations require different approaches. In a defend situation a very creative approach is required to escape. In an attack situation a very precise following of the opponent is required. It would be interesting to test different methods, like rule-based systems, decision-trees, or neural networks, for different situations.

Adaptive flight behavior

The current flight behavior is very static. The mathematical functions for the maneuver execution are only useful for one type of airplane, without problems. When, for example, the airplane is hit, by one single bullet in one of the wings, the agent will not be able to control the airplane anymore. To resolve this problem a method should be used for dynamic flight control. In [Liang, 2004] a neural network is used that can adapt during the actual flight. The results in that project are however far from usable in a dogfight situation.

Probabilistic approach

All data used in this project is concrete. There is no missing data and there is no uncertainty. This is, because the environment of the agent is a computer game in which all necessary data is available at all time. In a real life environment this would not be the case. In that case a probabilistic approach might be necessary. Such a real life situation could be emulated by introducing noise in the data that is retrieved from the MSCFS game. If one sees it as an end goal to create an agent that can deal with a computer game, a probabilistic approach will not be necessary. But when one wants to take a step further and extend an agent like the one described in this report to be able to deal with a real-life situation, it might be interesting to think of methods that can handle errors in the data.

Bibliography

[Andriambololona, 2003]
Andriambololona, M. and Lefeuvre, P. (2003). "*Implementing a dogfight artificial pilot*", Research report DKS03-07/ICE07, Knowledge Based Systems group, Delft University of Technology, The Netherlands.

[Capkova, 2002]
Capkova, I. Juza, M. and Zimmermann, K. (2002). "*Explorative data analysis of flight behavior with neural networks*", Research report DKS02-04/ACE02, Knowledge Based Systems group, Delft University of Technology, The Netherlands.

[Ehlert, 2003]
Ehlert, P.A.M. (2003). "*The Intelligent Cockpit Environment Project*", Research report DKS03-04/ICE04, Knowledge Based Systems group, Delft University of Technology, The Netherlands.

[Laird, 1998]
Laird, J.E. and Jones, R.M. (1998). "Building Advanced Autonomous AI Systems for Large Scale Real Time Simulations", in *Proceedings of the 1998 Computer Game Developers' Conference*, pp 365-378. Long Beach, California, USA.

[Lent, 1998]
Lent, M. van and Laird, J.E.(1998). "Learning by Observation in a Complex Domain", in *Proceedings of the Workshop on Knowledge Acquisition, Modeling, and Management*. Banff, Alberta, Canada.

[Liang, 2004]
Liang, Q. (2004) "*Neural flight control autopilot system"*, Research Report DKS04-04 / ICE 09, Knowledge Based Systems group, Delft University of Technology, The Netherlands.

[Machado, 2002]
Machado, R. (2002). "Rod Machado's Ground School*" Microsoft Flight Simulator 2002*.

[Mouthaan, 2003]
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.

[Russell, 1995]
Russell, S.J. Norvig, P. (1995). "*Artificial Intelligence, a Modern Approach*" Prentice Hall, Inc.

[Solinger, 2004]
Solinger, D. (2004). "*Creating a dogfight agent: a literature study"* , Research Report DKS04-03 / ICE 08, , Knowledge Based Systems group, Delft University of Technology, The Netherlands.

[Solinger, 2005]
Solinger, D. (2005). "*Creating a dogfight agent"* , Technical Report DKS05-01 / ICE 10, , Knowledge Based Systems group, Delft University of Technology, The Netherlands.

[Spronk, 2003]
Spronck, P. Sprinkhuizen-Kuyper, I. and Postma, E. (2003). "Improving Opponent Intelligence Through Offline Evolutionary Learning", in *Proceedings of the Fourteenth Belgium-Netherlands Conference on Artificial Intelligence*, pp 299-306.

[Tamerius, 2003]
Tamerius, M.S. (2003). "*Automating the cockpit: constructing an autonomous human-like flight bot in a simulated environment*", Technical report DKS-03-05/ICE05, Knowledge Based Systems group, Delft University of Technology, The Netherlands.

[Tozour, 2002]
Tozour, P.. (2002). "The Evolution of Game AI", in *AI Game Programming Wisdom*, Rabin, S. (editor) Charles River Media, Inc., pp 3-15.

[Virtanen, 2001]
Virtanen, K. Raivio, T. and Hämäläinen, R.P. (2001). "Modeling Pilot's Sequential Maneuvering Decisions by a Multistage Influence Diagram", in *Proceedings of the AIAA Guidance, Control, and Navigation Conference*, Montreal, Canada, 2001.

[Waveren, 2001]
Waveren, J.M.P. van (2001). "*The Quake III Arena Bot*", MSc. thesis, Knowledge Based Systems group, Delft University of Technology, The Netherlands.