Deutscher Luft- und Raumfahrtkongress 2012
DocumentID: 281398

AUTONOMOUS MISSION PLANNING FOR UAVS: A COGNITIVE APPROACH

A. Stenger, B. Fernando, M. Heni
SILVER ATENA Electronic Systems Engineering GmbH, Munich

Abstract

Autonomous operation is becoming an increasingly important factor for UAVs. It enables a vehicle to decide on the most appropriate action under consideration of the current vehicle and environment state. We investigated the decision-making process using the cognitive agent-based architecture *Soar*, which uses techniques adapted from human decision-making. Based on *Soar* an agent was developed which enables UAVs to autonomously make decisions and interact with a dynamic environment. One or more UAV agents were then tested in a simulation environment which has been developed using agent-based modelling. By simulating a dynamic environment, the capabilities of a UAV agent can be tested under defined conditions and additionally its behaviour can be visualised. The agent's abilities were demonstrated using a scenario consisting of a highly dynamic border-surveillance mission with multiple autonomous UAVs. We can show that the autonomous agents are able to execute the mission successfully and can react adaptively to unforeseen events. We conclude that using a cognitive architecture is a promising approach for modelling autonomous behaviour.

1. INTRODUCTION

1.1. From UAVs to Autonomous Agents

During the last century, UAVs have developed from mere flying bombs to complex systems which can serve a variety of different purposes. The benefits of these vehicles lie in the (comparatively) low production and operating costs and the flexibility to adapt the aircraft to the specific demands of its mission. However, the most obvious advantage is the absence of an on-board human pilot. This enables flying long-enduring missions or operations in hazardous conditions, often described by the term "3D-missions" (Dull, Dirty and Dangerous).

The desire to reduce human involvement leads to an increased demand for autonomously operating vehicles. Therefore, the development of such systems is expedited both for military and civil applications. Although the concept of autonomy is not new, it is hardly used in aeronautical applications – mainly due to legal and safety issues. Vehicles like the General Atomics RQ-1 Predator and the Northrop Grumman RQ-4 Global Hawk for example carry very advanced sensing systems that collect and transmit data. However, the aircraft themselves don't use that data to plan their missions autonomously - at least high-level decisions have to be made by a human operator. Therefore the potential of these aircraft is not fully utilised.

Automatic systems are an essential part of modern flight control systems and have significantly helped to increase the efficiency, comfort and safety of flight. However, the terms "automatic" and "autonomous" are often used synonymously, which can be misleading. We distinguish in the following: an automatic system is designed to fulfil a pre-programmed task. It cannot place its actions into the context of its environment and decide between different options. An autonomous system on the other hand has the capability to select amongst multiple possible action sequences in order to achieve its goals. The decision which action to choose is based on the current knowledge, that is, the current internal and external situation together with internally defined criteria and rules. In a dynamic world, there are usually many possible ways to achieve complex goals. The challenge is to find not just any solution to a problem, but a good or ideally the best one [1, p.1] [3, pp.141f].

Typically, the grade of control by a human operator as well as the implemented decision capability is directly related to the automatic or autonomous level of the system: with an increasing level of autonomy the decision freedom of the system rises and the grade of human intervention or interaction decreases [11, p.2]. For instance, in the area of flight controls, an automatic autopilot would be capable of following a predefined route. An autonomous autopilot however can choose the optimum route under consideration of the current situation of its environment, and then follow it.

A widely accepted metric on how to classify the grade of autonomy of an unmanned aerial vehicle is the Autonomous Control Levels (ACL) developed by the U.S. Air Force Research Laboratory (AFRL). The levels range from simple remotely piloted vehicles (level zero) to fully autonomous, human-like vehicles (level ten). In order to determine the degree of autonomy, several operational areas are taken into account, such as perception and situational awareness, analysis and coordination, and decision making [1, p.4].

1.2. Decision-making Systems

When talking about autonomous entities the term *agent* is frequently used. Stuart Russell and Peter Norvig define an agent as anything that can perceive and influence its environment by using sensors and actuators [10, p.34]. Following this general definition a sensor is any means of collecting information about the environment in which the agent acts. For a UAV, sensors can be air data sensors, position sensors or cameras. Similarly, an actuator is any means of affecting the environment of the agent.

The systems of an agent can be viewed on different levels of abstraction, as shown in FIGURE 1. On the lowest level the physical implementation of the system (i.e. computer, robot, UAV, etc.) is situated. It includes the hardware and the low-level software functions needed for basic computation tasks. The knowledge level, on the other end of the abstraction hierarchy, incorporates all the facts of the environment and the behavioural rules the agent needs to know in order to act according to its design. The knowledge base consists of both implemented and acquired knowledge. Furthermore, knowledge can be differentiated in categories such as general knowledge (which is used in general cognitive capabilities, such as language processing) and task knowledge (which is used in specific domains).

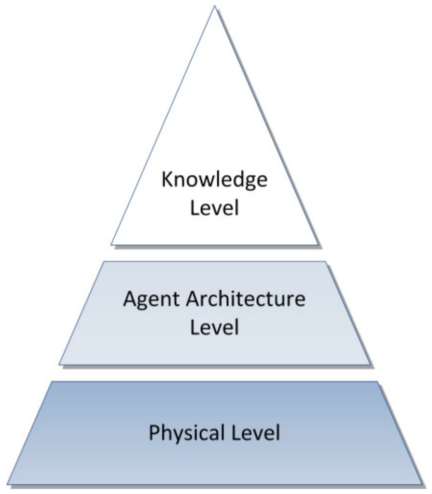


FIGURE 1. Abstraction hierarchy of an agent

Between the physical level and the knowledge level the agent architecture level is located. It forms the interface to the physical level and comprises the software structure of the agent as well as the routines and algorithms to acquire, represent and process knowledge. John E. Laird, one of the developers of *Soar*, summarises the working principle of an agent in the simple equation [8, pp.7f]:

architecture + knowledge = behaviour.

Several architectures have evolved as a result of different approaches to implement autonomy and from various operational fields for autonomous systems [9]. However, they can be summarised in the following basic categories:

- reactive architectures
- deliberate architectures
- layered or hybrid architectures

A reactive architecture uses condition-action rules to generate behaviour. It selects actions depending on its current perception, while ignoring the rest of its perception history. The great advantage of reactive agents is their capability of quick responses to dynamic changes in the environment. Additionally, there is no need to have a complete set of knowledge and data on the environment since the reaction is solely based on the current perception. However, it lacks the ability to reason about decisions, to reflect on long-term memory or to follow long-term goals. The behaviour of a reactive agent is event-driven. As a matter of fact, the line between autonomous reactive agents and purely automatic systems is not clearly distinguishable [12].

A deliberate architecture on the other hand comprises a symbolic representation of the world and uses logical reasoning to make decisions. In contrast to the reactive agents, the behaviour of the deliberate agent is usually goal-driven. This ability is advantageous especially with respect to autonomy. However, in order to make a good decision an accurate and fairly complete representation of the environment is required. In a complex world this is often not possible. Further difficulties arise when the knowledge base is incomplete or even false.

To combine the advantages of the reactive architectures (quick responses) with those of deliberate architectures (goal-driven reasoning), a layered architecture can be used. Layered architectures comprise at least two layers (reactive and deliberate) on different levels of abstraction.

Nevertheless, neither of the architectures really addresses the concept which the agent development originates from, namely cognitive modelling. When Allen Newell and Herbert Simon first formulated their pioneering work on problem solving in 1956, their intention was to phrase human problem solving skills [8, p.3]. Inspired by this work many approaches to simulate cognitive behaviour were developed. Using the research on psychological experiments conducted to understand human problem-solving techniques, researchers began to explore ways to solve problems artificially. However, the development of these deliberate architectures split up into planning systems and cognitive systems. Planning takes a symbolic representation of both the world and the states to find a set of actions to move from the present state to a goal state. Cognitive architectures also include deliberate principles and symbolic representations to make decisions. However, they also incorporate principles of reactive systems. What clearly distinguishes them from other architectures is the origin from human behaviour modelling.

One of the oldest and most developed cognitive architectures is *Soar*. It was first introduced by Laird and Newell in 1983 and has been continually developed [8, p.10]. The intended goal of the *Soar* project is nothing less than to recreate the full scope of human-like behaviour, from basic routine actions to complex problem solving tasks. Current versions comprise several methods and techniques to enable both reactive and deliberate behaviour.

2. THE AUTONOMOUS AGENT

In the following we will describe the autonomous agent of a UAV, without going into detail about the physical properties and the performance of the vehicle itself. We propose to implement autonomy as a layered agent architecture that uses a hierarchical structure with a deliberate planner on the top and the cognitive architecture *Soar* on the bottom (see FIGURE 2).

In order to generate plans in a fully observable environment a deliberate planner is used. Such a planner includes search algorithms that are capable of finding not only a solution that is accurate and correct, but which is good or the best in terms of the given state and goal description [6]. The resulting plan concerns the top-level decisions that are necessary to generate an overall mission application. At the bottom level, *Soar* is responsible for handling all low-level decisions and in particular for reacting to unexpected events. At this level, purely deliberate behaviour would be disadvantageous due to frequently occurring unexpected events combined with limited knowledge about every detail on the environment which will lead to unknown world and system states. Even

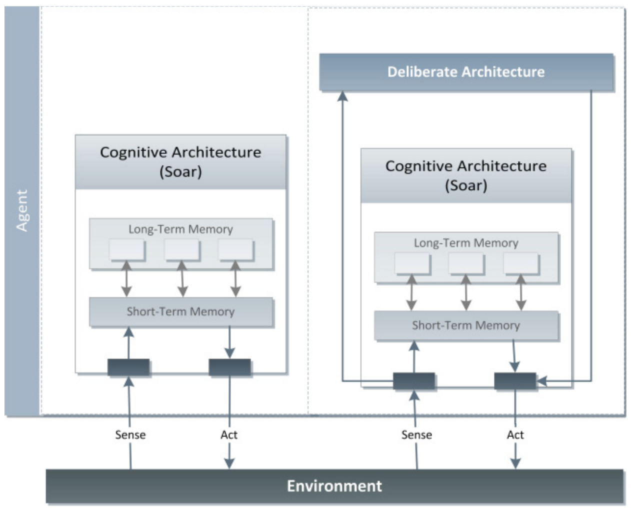


FIGURE 2. Left: Agent based solely on *Soar*; Right: layered agent architecture

when using sophisticated plan monitors and re-planning capabilities this easily invalidates overly detailed deliberate plans. The consequence is frequent re-planning and thus a sub-optimal overall goal achievement. The *Soar* agent is better suited for making reasonable decisions in a complex and dynamic environment, such as in a UAV domain, where sometimes only limited or incomplete knowledge is available. Since the *Soar* cognitive architecture itself is a hybrid architecture, it integrates both the abilities to react to certain situations and to deliberately reason about a decision. In addition, it integrates the capability to use its percept history and to learn from impasse resolutions that occurred from a lack of knowledge. Although *Soar* is well-suited for low-level decisions that require fast decision-making, it will not necessarily find an optimal solution.

In the following, we will focus only on the *Soar* part of the agent and describe its capabilities in more detail. The deliberate planner is discussed elsewhere [6].

2.1. *Soar* System

In order to generate behaviour, an agent requires encoded knowledge. In *Soar*, knowledge is encoded using so-called *working memory elements* (WMEs), which can hold any piece of information. For that, a WME has an identifier, an attribute, and a value. In case a WME contains several attribute-value pairs that share the same identifier, it is called an object, as depicted in FIGURE 3.

```
(<aircraft> ^altitude 3000              object <aircraft>
            ^speed 120
            ^position <position>)

(<position> ^latitude 48.0              object <position>
            ^longitude 11.5)
```

FIGURE 3. Working memory element structures

However, working memory alone generates no behaviour. The structures that store the WMEs and the operators that change a state are located in the long-term memory. Similar to human cognition, there are three types of long-term memory in *Soar*: *procedural*, *semantic*, and *episodic memory* (see FIGURE 4).

Procedural memory contains the information from working memory, which is stored in *productions* or *rules*. In *Soar*, a production always contains a condition and an action side. The conditions are responsible for checking whether they match the current state, and if the rule can fire.

In order to decide which rule to take, *Soar* uses a decision cycle that includes the following four phases:

- *elaboration*: Explicit mapping of the current state
- *proposal*: Propose appropriate operators that match the current state and compare them
- *decision*: Operators are selected based on their preferences, the results from the comparison, or the result from sub-goaling due to an impasse resolution
- *application*: Productions fire to apply the operator, modifying the current state

While executing the proposed applicable operators, their conditions have to be matched against the state. Assuming a fairly simple problem space with 10 elements in working memory (W), 20 available productions (P), with each holding in average 5 conditions (C), this would already result in 10^100[1] necessary comparisons. In order to avoid this escalating amount, *Soar* uses the *Rete-algorithm* [8, p.23]. Instead of matching each condition to the working memory, *Rete* compares only the changes and stores partial matches additionally.

Finding an appropriate match and thus a decision right away depends on the available knowledge. If *Soar* is unable to make progress during the decision cycle, it reaches an impasse. *Soar* recognizes four types of impasses:

- *state no-change impasse*: no preference in working memory
- *tie impasse*: multiple operators with equal preferences
- *operator no-change*: operator remains selected
- *conflict impasse*: multiple operators with conflicting preferences

Soar resolves an impasse by using *sub-goaling*. The response is the creation of a sub-state, which holds the reason for the impasse and a mapping of the state. As soon as a solution can be found, a production rule (*chunk*) is created that contains the condition that caused the impasse and the action that solved it. Consequently, working memory elements can be transferred to procedural long-term memory by *chunking*. To prevent repetitive sub-goaling, those chunks can be stored when learning is enabled.

Another method to transfer information is *reinforcement learning* [8, pp.181f], which is implemented in recent versions of *Soar*. Reinforcement learning differs from the general learning mechanism in such a way that it also includes a reward-link. Every time the *Soar* agent comes across a numerically indifferent preference, it changes the numeric value, based on the generated reward. An agent hence learns to choose those operators that lead to the most promising reward by trial and error.

Recent versions of *Soar* also support the capability to use semantic memory, which can be expanded and retrieved by using declarative chunks (semantic learning). In contrast to procedural memory they hold the general

[1] W^(C*P) = 10^(5*20) = 10^100

knowledge, which does not need to be acquired by learning. Semantic memory forms the agent's knowledge about the world. Unlike procedural memory, however, it is neither stored as rules nor just in working memory, but as simple facts.

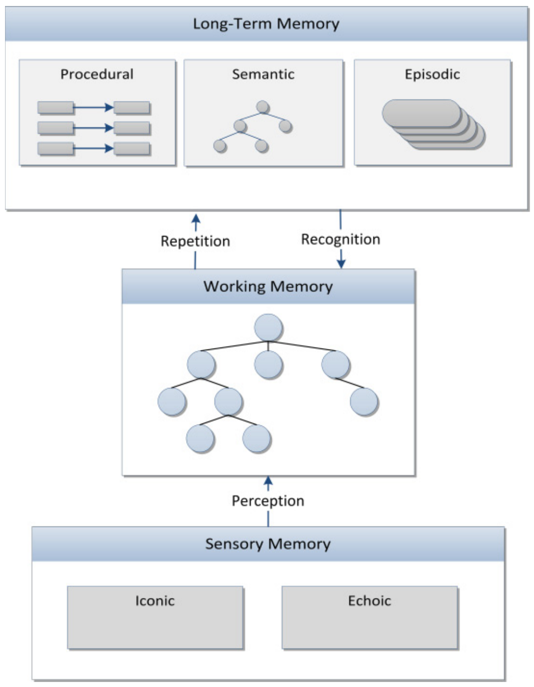


FIGURE 4. *Soar* memory structure

Another recent feature in *Soar* includes episodic memory, the third type of long-term memory. Using this, entire snapshots of working memory (episodes) can be created, which enable the agent to store specific case-based problem solutions and to remember critical situations that have occurred in the past.

3. MISSION SCENARIO

Within this article we describe an autonomous UAV agent that performs a mission in the context of a border surveillance application. Deploying UAVs for this task is favourable due to their long endurance and vast area coverage. This scenario is also of particular interest because on the one hand it is a potential application for real-life civil UAVs and on the other hand it is well-suited to evaluate the reliability of an autonomous agent.

Using an agent-based simulation framework (see section 4), the UAV agent will be exposed first to a nominal mission, where few unexpected events occur and second to a stressed mission, where the original planned mission will be invalid due to frequently occurring unforeseen events. Especially in the second case, the autonomous agent can demonstrate its capabilities to make decisions even with incomplete knowledge.

3.1. Nominal Mission

This scenario requires one or more UAVs to patrol a designated border region and to detect intruders (see FIGURE 5). Additionally it contains a ground-based (un)manned control station (GCS) that is responsible for managing the overall mission aspects. The GCS also acquires planning tasks that are executed by the deliberate architecture like issuing a detailed mission description or managing the UAV deployment. As such, it has access to all UAVs which are principally assigned for the mission and which are in one of the following four states:

- *available*: The UAV is currently deployed for the mission. It is airborne and exhibits no system errors.
- *stand-by*: The UAV is capable of being used for the mission, but is currently not. It is either airborne or on ground and exhibits no system errors.
- *engaged*: The UAV is temporarily not available for the mission, because it is engaged with another task. It exhibits no system errors.
- *unavailable:* The UAV is incapable and cannot be used for the mission due to failures.

In the event that more UAVs are required to take part in the mission, one UAV in *stand-by* will receive the mission plan. The aircraft state then changes to *available* and the UAV starts following the assigned tasks.

The primary task will engage the UAV to fly to the search area. Within the search area the UAV starts scanning the border region by using an appropriate search strategy which depends on the sensor coverage and on the probability that intruders will cross the border at a specific region (with reference to the landscape and previously encountered violation occurrences).

As soon as a UAV detects an intruder or a group of intruders, it transmits its current position, as well as live video images to the GCS. The GCS then decides whether the UAV shall further track the intruder or whether it shall search within surrounding areas for additional violations, due to a false alarm (detected object is not an intruder).

At the same time, the GCS notifies the local border patrol of the detected violation by delivering the aircraft's position coordinates, its path, as well as real-time imagery. Depending on the GCS's decision of whether the UAV shall track the object or not, the UAV will change its status to *engaged* or will continue searching (state: *available*). The surrounding UAVs will then cover the search area of the preoccupied UAV until manned ground units take over or the intruder vanishes.

3.2. Stressed Mission

A nominal mission is unlikely to occur in reality. Consequently, the robustness of the agent is tested when confronted which typical stressors such as system failures or other aerial traffic. A selection of stressors that were applied to test the reliability of the agent and the suitability to use the *Soar* cognitive architecture include the following:

- *Moderate System Failure*
  A moderate system failure is a system failure that does not interfere with the aircraft maintaining a safe flight condition, but which hinders the resumption of the mission (e.g. loss of camera signals).
- *Close Traffic*
  In order to avoid a Traffic Alert and Collision Avoidance System (TCAS) violation or even a mid-air collision, the agent is temporarily required to interrupt the mission until it senses no traffic in proximity anymore.
- *Adverse Weather*
  Minor fog, high humidity or clouds, for instance, hinder the mission execution because the UAV is unable to detect and follow intruders due to poor camera

image quality. Furthermore, strong winds or hail can affect the safety of flight itself.

- *Critical Level of Resources*
  A critical level of necessary resources such as low fuel also prevents the agent from continuing the mission.
- *Multiple Intruder Detection*
  When observing an intruder an additional intruder appears, this might clobber its goal to follow the previously detected intruder.
- *Unavailable Landing Place*
  A UAV decides to land at an airport for refuelling. While approaching the area, it recognizes that the landing place became unavailable. The level of difficulty to find an appropriate solution rises when the amount of fuel already reached a critical level.

Besides confronting a UAV with a single one of these stressors, the level of difficulty for the agent to find a reasonable decision further increase when multiple stressors are combined at the same time. The autonomous agent is then analysed by its capability to find a solution, especially when more than one option is available.

3.3. Mission Complexity

Mission complexity is a key factor with respect to a potential scenario. It is classified into three categories: *mission*, *team*, and *communication management*. *Mission management* describes the ability for a given mission to have a high amount of diverse capabilities and goals. This gives the agent the opportunity to act upon a broad number of possibilities. Hence, the mission puts no or few constraints on the UAV's capabilities. *Team management*, on the other hand, describes to which extent members or agents of the team are able to cooperate and to maintain themselves. The capability for them to communicate and perhaps start a formation within the mission is specified in *communication management*.

Based on those definitions, the prevailing mission fulfils the following requirements and thus results in a mission that is complex enough to test the agent's capabilities:

- *Mission Management:* high amount of goals and mission specifications
- *Team Management:* need for cooperation given
- *Communication Management:* need for formation control and/or ATC given

In order to classify the degree of autonomy required by this mission within the Autonomous Control Levels (ACL), we need to consider the following aspects. The chosen scenario allows for a moderate level of communication and cooperation between the UAVs, thus increasing the independence from human intervention significantly. Considering the scope and complexity of the mission an adequate prediction capability as well as real-time analysis and decision-making are required. The mission environment can be considered challenging and thus demands an advanced level of perception and situational awareness. Therefore, the ACL ranges between four (communication, cooperation, and decision-making) and five (perception and situational awareness).

4. SIMULATION ENVIRONMENT

Simulation is a significant aspect in every modern development process. It allows optimising the developed system with regard to its future task area in an early stage of

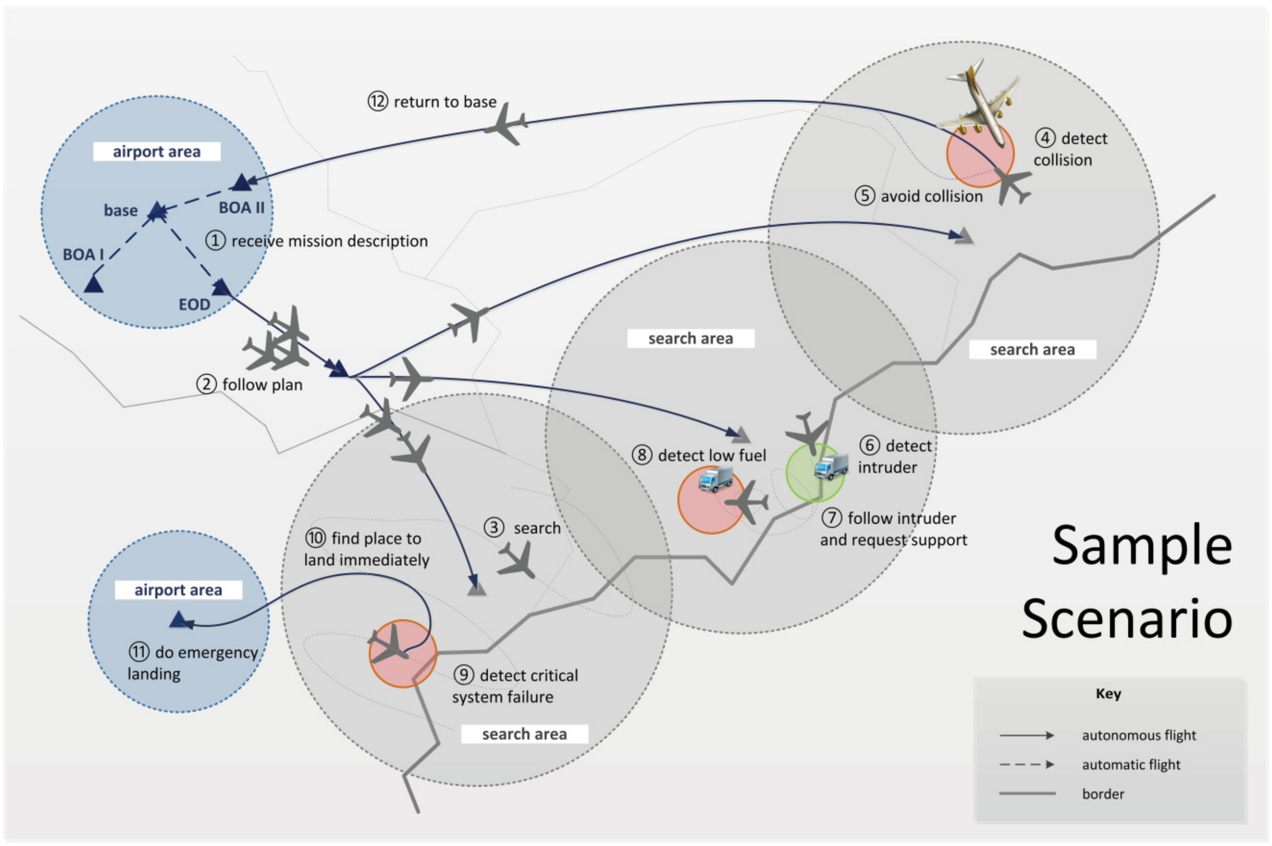


FIGURE 5. Stressed sample mission, where the numbers indicate the mission sequence.

the project and helps to cut project costs. Therefore, developing a comprehensive simulation framework was considered a substantiate part within the development of the autonomous agent. In order to validate the behaviour of the developed autonomous system, the simulation environment provides the decision-finding system with all relevant data and simulates the execution of the generated action plans.

A promising approach in modern software development is the concept of agent-based modelling and simulation. Since software agents have a lot in common with autonomously operating UAVs, it seems plausible to realise the simulation environment in the context of an agent-based simulation platform. For this purpose, the agent-based modelling toolkit A-Globe was used. It was developed by the Agent Technology Group at the Czech Technical University in Prague under sponsorship of the U.S. Air Force. [1, p.1]

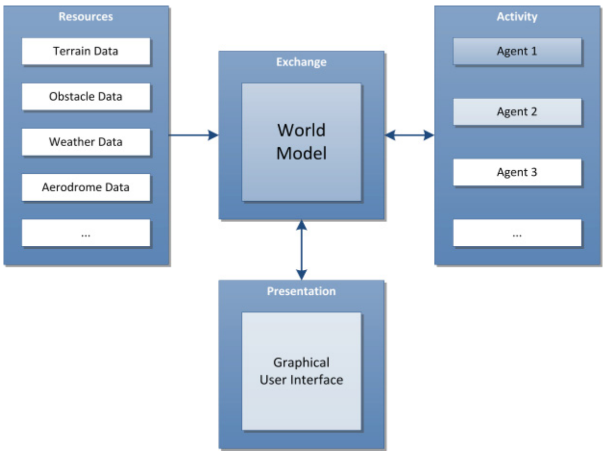


FIGURE 6. Simulation environment architecture

One of the main design considerations in the development of the simulation environment was modularity. In view of this, the architecture is divided into the functional areas of *Resources*, *Activity*, *Exchange* and *Presentation*, as depicted in FIGURE 6. The area of *Resources* consists of data which form the basis of the physical world and which are contained in external files. The field of *Activity* is made up of the autonomous agents who possess the ability to perceive the current state of the world and their fellow agents and plan their actions accordingly. Every agent's state is simulated independently, thus the simulation operates on a rather decentralised level. FIGURE 7 depicts the simulation loop as well as the sensor and effector chains within the agent.

The central component of the simulation framework is the World Model, which forms the area of *Exchange*. By generating an internal representation of the physical world, a mission environment is created in which autonomous entities can exist and operate. The World Model consists of a database which stores information about terrain, weather and objects in the world, such as airports or obstacles. It also holds the states of all simulated agents, in order to allow interaction between them.

The connection to the decision-finding system is established via a generic bus simulation. The sensor data is abstracted by using a universal message catalogue which has an xml-based standardised format defining the name, range and accuracy of the transmitted data and the position of the signal on the bus. The decision-finding system uses the same message catalogue to decode the bus signals and convert the data into physical values. This interface assures that several autonomous systems can be easily linked with the rest of the agent simulation and can be compared to each other.

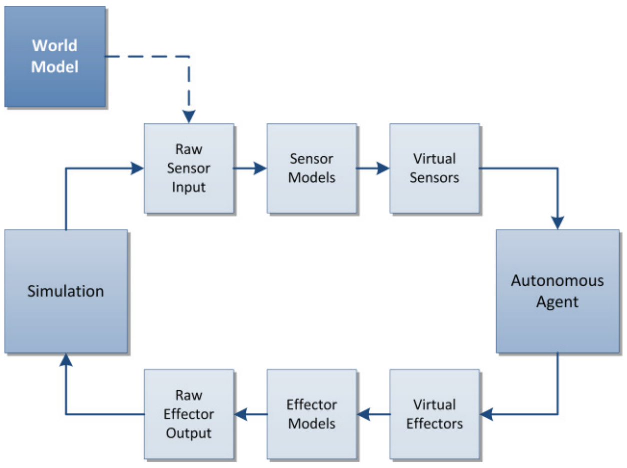
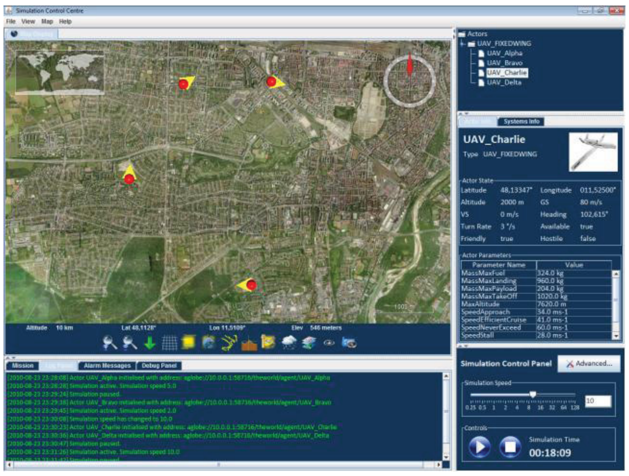


FIGURE 7. The agent's inner simulation loop

The simulated autonomous agents can be equipped with different levels of autonomy, which may range from pure reactive bearing, following simple rules to highly deliberate behaviour as a sequence of complex actions. The generated action plan of the decision-finding system is carried out and the activities of all simulation participants are visualised and recorded. This gives the developer the opportunity to monitor, validate and further optimise the functionalities of the autonomous system. For the simulation of the UAV's motion, a linear four DOF flight dynamics model is employed. This is derived from a six DOF flight dynamics model by restricting pitching and rolling motion as well as neglecting any sideslip. This simplification was made to receive a flight dynamics model with sufficient accuracy to simulate autonomous agents while at the same time avoiding the effort to develop a comprehensive flight dynamics model. Consequently, the focus lies on the accurate simulation of autonomous behaviour rather than detailed motion.



UAV_Charlie
Type UAV_FIXEDWING

Actor State

Latitude	48,13347°	Longitude	011,52500°
Altitude	2000 m	GS	80 m/s
VS	0 m/s	Heading	102,615°
Turn Rate	3 °/s	Available	true
Friendly	true	Hostile	false

Actor Parameters

Parameter Name	Value
MassMaxFuel	324.0 kg
MassMaxLanding	960.0 kg
MassMaxPayload	204.0 kg
MassMaxTakeOff	1020.0 kg
MaxAltitude	7620.0 m
SpeedApproach	34.0 ms-1
SpeedEfficientCruise	41.0 ms-1
SpeedNeverExceed	60.0 ms-1
SpeedStall	28.0 ms-1

Altitude 10 km Lat 48,1128° Lon 11,5109° Elev 546 meters

Simulation Speed 10

Simulation Time 00:18:09

```
[2010-08-23 23:28:08] Actor UAV_Alpha initialised with address: aglobe://10.0.0.1:58716/theworld/agent/UAV_Alpha
[2010-08-23 23:28:28] Simulation active. Simulation speed 5.0
[2010-08-23 23:29:24] Simulation paused.
[2010-08-23 23:29:38] Actor UAV_Bravo initialised with address: aglobe://10.0.0.1:58716/theworld/agent/UAV_Bravo
[2010-08-23 23:29:45] Simulation active. Simulation speed 2.0
[2010-08-23 23:30:08] Simulation speed has changed to 10.0
[2010-08-23 23:30:23] Actor UAV_Charlie initialised with address: aglobe://10.0.0.1:58716/theworld/agent/UAV_Charlie
[2010-08-23 23:30:36] Actor UAV_Delta initialised with address: aglobe://10.0.0.1:58716/theworld/agent/UAV_Delta
[2010-08-23 23:30:47] Simulation paused.
[2010-08-23 23:31:26] Simulation active. Simulation speed 10.0
```

FIGURE 8. The GUI of the simulation environment

Finally, the aspect of *Presentation* includes a graphical user interface (GUI, see FIGURE 8) which fulfils two important functions. On the one hand, it serves as the primary means for illustrating the simulation activity and thus gaining insight into the mode of action of the autonomous decision-making system. On the other hand, it offers the operator the possibility to interact with the simulation by controlling the simulation speed, making live modifications to the mission environment or creating new agents. Mission simulations can be automated by scripting a course of events. Further functions allow the analysis of internal procedures of the planner and the creation of arbitrary mission scenarios using an interactive mission designer as well as recording and replaying all events into/from a database or file.

5. IMPLEMENTATION

Most agent architectures are well-suited to perform a certain type of task, such as route planning or task decomposition, whereas they fail on tasks which they are not intended for. Even the *Soar* cognitive architecture is rather meant for modelling human cognition than for providing a complete architecture for unmanned vehicles. Due to its cognition model it is suitable for a wide range of applications such as simulating aircraft pilots in combat [5], simulating opponents in a video game [7], or learning from instructions [4]. For the application in a real UAV scenario however, using solely a cognitive agent architecture is difficult. Therefore, the proposed UAV agent has a two-level layered architecture where the *Soar* module's task is to handle the en-route short-term decisions.

The following sections provide an overview of the *Soar* component of the UAV agent and the results from the mission application. The deliberate planner is discussed in a different work [6].

5.1. *Soar* Module

The *Soar* cognitive architecture already provides general knowledge that is necessary for processing information, selecting suitable operators, resolving impasses, and for storing the experience from the interaction with the environment. We further define three types of productions that are stored as production rules in procedural memory and that integrate the knowledge to generate the behaviour in the UAV domain:

- system-specific knowledge (UAV)
- flight-specific knowledge (general manoeuvres)
- mission-specific knowledge (border surveillance)

System-specific knowledge concerns those productions that are explicit for the use in a UAV. Those rules are mainly pre-defined and include systems such as the UAV's sensors and actuators, containing, for instance, the rules that detect an unintended engine shut-down.

One solution for an engine failure could be to perform an emergency landing. The rules to accomplish this kind of task are defined within the *flight-specific* productions. They concern the aircraft's movement and flight-specific manoeuvres, including aerial refuelling and collision avoidance. It is important to note that none of these productions replace an existing aircraft system such as the Traffic Alert and Collision Avoidance System (TCAS), the Flight Management System (FMS) or the weather radar. Instead, they extend the scope of these systems by, for instance, enabling collision avoidance even with objects that are not equipped with a transponder (such as gliders or terrain). Furthermore, the obstacle recognition need not be limited to the TCAS range.

The productions that are necessary to accomplish a mission, however, are defined using *mission-specific* knowledge. In contrast to the other production types, those rules are only pre-defined to a certain extent. Consequently, they will develop further while interacting with the environment to improve the agent's decision-making.

As outlined in FIGURE 9, each type of knowledge can be further sub-divided into problem spaces. A problem space contains those productions that belong together such as *Communication*, *Team*, or *Mission*. However, there are also productions that are not tied to a problem space like *Flying*, *Searching*, and *Intercepting*. Those productions are necessary to provide tasks that can be performed at any time.

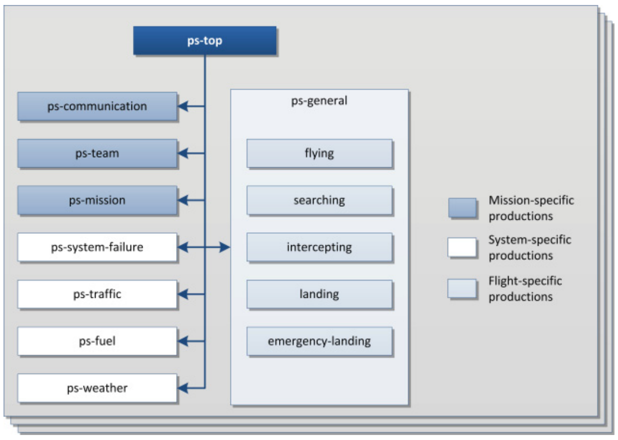


FIGURE 9. Selection of problem spaces

In order to decide which problem space has a higher priority, a top problem space (`ps-top`) is defined which contains the rules to prioritise. For instance, in general it is more appropriate to terminate the mission execution, if refuelling is necessary. Therefore, the problem space for refuelling (`ps-refuel`) has a higher priority than the one for searching (`ps-search`). In another occasion, however, this decision is rather inappropriate. If, for instance, there are only few or no UAVs available that can cover the search area and the aircraft has still sufficient fuel to reach more than one refuel place, `ps-search` would have a higher priority than `ps-refuel`.

5.2. Mission Application

To test the reliability of the agent it was exposed to several scenarios in the simulation framework including different levels of stressors. First, the decision-making capabilities are analysed when detecting an intruder in a simulation that contains two available UAVs. The difficulty here does not concern the confrontation with multiple stressors, but rather the reliability of the decision itself. As depicted in FIGURE 10 there is more than one appropriate rule available. In order to find the most suitable one (based on its current knowledge), the agent uses sub-goaling. Consequently, the UAV agent (`uav-1`) decides to follow the intruder, which results in the second UAV expanding its search area. While following the intruder, `uav-1` recognises after five[2] additional decision cycles that it is incapable of just following the intruder without changing its pursuit strategy.

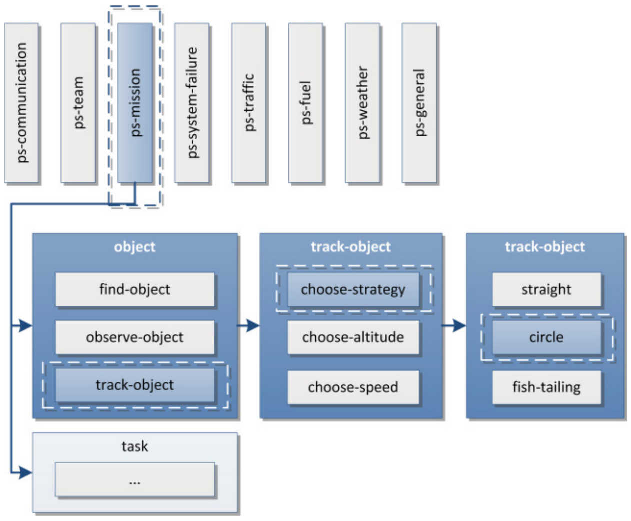


FIGURE 10. Multi-option reliability

The decision tree in FIGURE 10 also outlines that the UAV decides to follow the intruder by making circles and starts reflecting the decisions whether it is appropriate. For analysing the strategy the agent uses reinforcement learning, which enables it to become more accurate with every time that it is confronted to use a pursuit strategy for following a certain intruder.

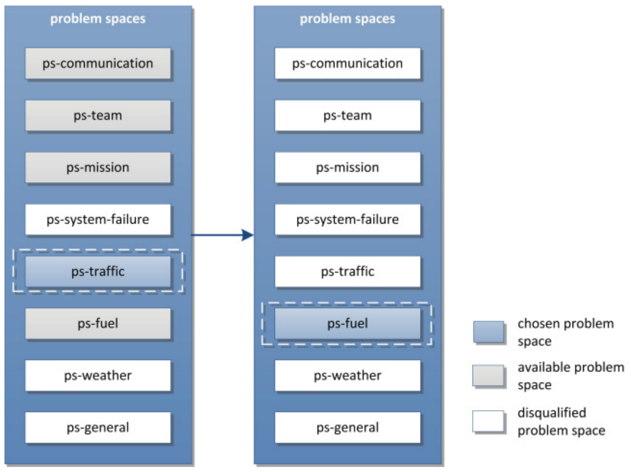


FIGURE 11. Multi-stressor reliability

In a second simulation, the reliability of the agent is tested when being confronted with multiple stressors. Again, the simulation starts with two UAVs, which search for intruders in a given area. Three stressors are now applied at the same time for `uav-1`: first, an intruder appears, second, another airborne object forces the UAV to avoid a collision, and third, the fuel amount drops below the safe level. Each of those stressors alone obliges the agent to make several decisions. Combining them increases the stress level, especially as this case is not pre-programmed.

An overview of the sequence of problem spaces which are entered in order to satisfy the top-level goal (avoiding a collision with any object at any time) is shown in FIGURE 11. In the scenario simulation, the agent decides to avoid the collision immediately, even though it was not yet imminent, and subsequently loses track of the intruder. This behaviour is not optimal and driven by the primary goal to avoid any collision. An improved behaviour can therefore be achieved by choosing a more indirect primary goal, namely for the agent to survive. Exposing the agent with the new primary goal `survive`, instead of `avoiding-collision`, the simulation with three stressors results in the desired behaviour: the agent first requests support from the second UAV and later on avoids the collision before it finally returns to the base for refuelling.

Applied to the mission as described in section 3 these aspects can be summarised in the following conclusions. Since the autonomous agent was designed for the nominal mission, this scenario is always resolved correctly. The application of stressors reveals the strengths and weaknesses of the *Soar* agent. If the mission contains only one or few stressors, this does generally not pose a problem to the system. However, if more stressors are applied, the performance deteriorates, leading to decisions which are feasible, but sometimes unreasonable. This behaviour depends not only on the number of stressors, but rather on their complexity and the temporal density of their occurrences. While *adverse weather* and a *critical level of resources* are comparatively simple stressors, *multiple intruder detection* and *close traffic* require a great number of inner decision and elaboration cycles. Furthermore, the correct choice of the top problem space is crucial to yield a good decision due to the changed processing order. For instance, the selection of the top problem space `survive` leads to a significantly more favourable result. Therefore, by choosing a proper primary goal which in particular is not too explicit, it is possible to address all stressors correctly.

5.3. Simulation results

Situations that require fast deliberate decision-making such as avoiding a collision are well-qualified for learning. Using look-ahead learning, for instance, the agent will always find the correct solution within a single decision cycle. After being confronted with the situation and solving it for the first time, the agent stores the decision in a chunk (as long as learning is enabled). This relationship is depicted in FIGURE 12. However, assuming that the situation changes partly (as simulated in the seventh run), the agent enters an increased amount of elaboration cycles, which results in an increased amount of production firings as well. The diagram also outlines that once a decision was stored in a chunk and the situation reoccurs, the agent will find the best decision right away again.

Storing each decision and analysing each path that leads to the desired state results in the best solution, but also in a performance decrease. Thus, using look-ahead planning is only appropriate when the situation necessitates optimal behaviour and if limitations regarding the hardware and the performance are acceptable.

Apart from look-ahead planning, another suitable way of learning during a collision avoidance scenario is rein-

[2] The number of time steps to track an object (such as an intruder or a manned aircraft) is set to five, in order to achieve a probable path extrapolation outcome.

forcement learning (RL). In RL, an agent learns the best policy (in this case the best sequence of actions), based on reward returned from the environment. The advantage of reinforcement learning compared to look-ahead planning is that in order to find the best policy, the agent uses a cumulated reward. This means that the agent considers that even if a present decision results in a positive reward due to its former experience, the decision will not always be the best one, because the situation might differ. FIGURE 13 illustrates that within the same situation, the number of decisions reaches its final low after four runs. The same is true for production firings and elaboration cycles, since they correlate. In this case, the partly changed initial conditions result only in a slight increase in elaboration cycles and production firings. Compared to look-ahead planning, RL is well-integrated in Soar, having nearly no impact on the performance. Consequently, RL is a good measure for increasing an agent's overall performance, especially in situations that do not necessitate optimal behaviour, but which require fast deliberate decision-making.

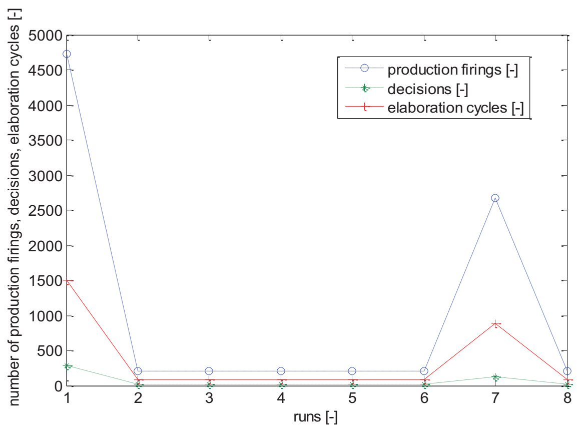


FIGURE 12. Number of production firings, decisions, and elaboration cycles with look-ahead learning

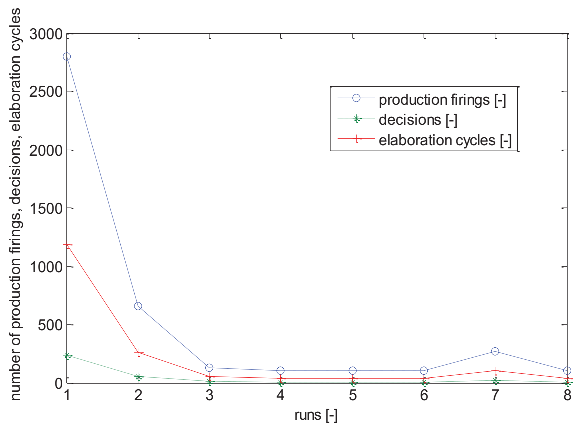


FIGURE 13. Number of production firings, decisions, and elaboration cycles with reinforcement learning

5.4. Suitability of *Soar*

Using the border surveillance mission, it could be shown that *Soar* is capable of making detailed decisions even during unforeseen events. Its main strength is that on the one side it differs from pure reactive systems which simply react to the present situation without further consideration of the environment or the future. On the other side, it is also better suited than a pure deliberate system because it is more flexible when confronted with unforeseen events.

The main advantage of *Soar*, however, is its ability to form complex behaviour from the interaction with the environment and not by following pre-programmed action sequences. Instead, it finds solutions by forming new operators.

Still, *Soar* is not a planning system and therefore not as well-suited as a pure planner [6] for finding an optimal solution for an overall mission plan. When the learning components of *Soar* are enabled, the *Soar* agent adapts and improves over time by using its experience. Therefore, it offers various methods for deducing a learning strategy and for learning solutions (reinforcement learning, semantic learning, episodic learning, chunking). Although each of these strategies aims at storing experience in long-term memory, they differ in the time of their appearance and the type of their implementation. Whilst episodic learning and chunking will serve to reduce the search time, reinforcement learning aims at improving the solution.

6. SUMMARY

Unmanned aircraft are deployed in domains that humans are not active in due to safety issues or an adverse mission environment. However, UAVs are not supposed to work completely autonomously without any human involvement, be it spontaneous intervention, remote control or simply provision of the mission objectives. It is thus not desired or required for a UAV to be completely autonomous in terms of its overall deployment. Nevertheless, a higher degree of autonomy allows a lower workload for human operators or enables a human operator to control several UAVs at the same time. More autonomy is also useful in the event of limited or lost communication as it allows the UAV to continue its mission alone instead of requiring a premature return to base.

Flight safety is a critical aspect for UAV usage as aviation is more focused on low accident rates and consequently on a high safety rate than any other means of transportation. Consequently, increased autonomy especially with systems such as *Soar,* where the outcome of a decision is not always easily predictable, are difficult to use in aviation. There are two possible scenarios how to benefit from a cognitive architecture but at the same time maintain safe flight. First, the cognitive architecture is clearly separated from any safety critical aspects of the system. No decision of the *Soar* agent may interfere with a flight-critical aspect of the system. Second, safety-critical monitors are implemented which verify any decision made by the cognitive architecture. Decisions are rejected if they contradict a safe flight. For example control commands outside the flight envelope or commands violating TCAS orders are rejected. The system then falls back to the pre-programmed mode which handles such a situation as any system without autonomy would.

The agent-based simulation environment has proven to be a valuable tool for evaluating the function of the autonomous agents during the development process. All motion and behaviour of the agents and the world can be simulated and test missions can be designed rather freely. Thus, any of the required traffic situations, intruders or obstacles can be simulated, including the reaction of the agent. A defined mission setup and a scripted course of events help recreating identical situations to compare different versions of the autonomous agent. Additionally, an intuitively operable graphical user interface facilitates analysing and understanding the behaviour of the simulated agent. The modular design of the simulation environment supports easy maintenance and makes it possible to expand it as the need arises.

The proposed UAV agent uses the advantages of a layered architecture where a high-level planner generates an overall mission plan and the cognitive architecture *Soar* is used for en-route decisions and reactions to unforeseen events. High-level planners perform very well on complex route and mission planning problems in a well-known environment. However, one major aspect when deploying UAVs for patrolling a border region is the concern of a limited situational awareness and a constantly changing environment. In this case the *Soar* architecture can show its strength by handling unexpected events and coping with a just partially known environment.

Soar performs very well on such an intermediate level, located between a reactive or pre-programmed system and a deliberate planner. In particular, it provides more autonomy than a purely reactive system. Still, *Soar* is not a deliberate planner which would yield an optimal plan (although such an implementation is in principle possible), therefore it is not used for high-level mission plans.

To increase flexibility, the cognitive part of the proposed agent includes only a small amount of pre-programmed task sequences. The majority of complex actions evolve from both interaction with the environment and innate learning mechanisms. The simulation demonstrates that the use of learning mechanisms can be advantageous as long as it is reduced to individual problem spaces or subtasks and the agent is given the ability to unlearn behaviour as well. Parts of the learning process are problematic in real UAV applications where decisions should be right the very first time. However, it is possible to expose the agent to typical scenarios in a simulation environment where part of the learning process can already be undergone. Other parts of the learning (e.g. chunking) are not critical as they only serve to improve performance and can be done at any time during a mission.

With a growing demand for the deployment of UAVs in all types of civil and military applications the necessity for more autonomy will also increase. In both the nominal as well as the stressed mission the *Soar* agent was able to handle all applied stressors successfully. We conclude that the proposed concept for using *Soar* as one part of a layered architecture of an autonomous UAV agent is a promising approach for equipping the vehicle with more autonomy.

ACKNOWLEDGEMENTS

This work was supported by SILVER ATENA Electronic Systems Engineering GmbH in Munich, Germany. The authors would also like to thank FH-Prof. Dipl.-Ing. Dr. Holger Flühr and Dipl.-Ing. Dr. Bernd Messnarz of the Department of Aviation at the FH JOANNEUM University of Applied Sciences in Graz, Austria, for their valuable contributions and comments.

REFERENCES

1 Agent Technology Center, “A-Globe Manual," [online paper], URL: http://agents.felk.cvut.cz/download/aglobe/aglobe-manual.pdf [cited 23 Jun 2010].

2 Clough, B. T., “Metrics, Schmetrics! How the Heck Do You Determine A UAV's Autonomy Anyway?" in *Proceedings of the 1st AIAA Conference on Unmanned Air Systems*, No. 990, Gaithersburg, Maryland, 2002, pp. 313-319, URL: http://www.cs.uml.edu/~holly/91.549/readings/clough-permis02.pdf [cited 7 June 2011].

3 Gunderson, J. P. and Gunderson, L. F., “Autonomy (What's it Good for?)," *PerMIS '07: Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, ACM, New York, NY, 2007, pp. 141-147.

4 Huffman, S. B. and Laird, J. E., “Learning from instruction: A knowledge-level capability within a unified theory of cognition,” *Proceedings of the Fifteenth annual Conference of the Cognitive Science Society*, 1993.

5 Jones, R. M., Tambe, M., Laird, J. E., and Rosenbloom, “Intelligent automated agents for flight training simulators,” *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, P.S., Orlando, FL, 1993.

6 Kreinbucher, B., “Autonome Entscheidungsfindung für unbemannte Luftfahrzeuge unter Verwendung von hierarchischer Aktionsplanung,” Diploma thesis, FH Joanneum University of Applied Sciences, Graz, 2010.

7 Laird, J. E., “It Knows What You’re Going To Do: Adding Anticipation to a Quakebot,” [online paper], Ann Arbor, Michigan, 2001, URL: http://ai.eecs.umich.edu/people/laird/papers/Agents01.pdf [cited 29 June 2012]

8 Laird, J. E., *The Soar Cognitive Architecture*, MIT Press, Cambridge, Massachusetts, London, England, 2012.

9 Müller, J. P., “The Right Agent (Architecture) to Do the Right Thing,” in *Intelligent Agents V: Agent Theories, Architectures, and Languages*, edited by J. P. Müller, A. S. Rao, and M. P. Singh, Vol. 1555 of Lecture Notes in Computer Science, *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Heidelberg, New York, July 1999, pp. 211-242.

10 Russell, S. J. and Norvig, P., *Artificial intelligence: A modern approach*, 3rd ed., Prentice Hall Series in Artificial Intelligence, Prentice Hall, Upper Saddle River, New Jersey, 2010.

11 The Royal Academy of Engineering, „Autonomous Systems: Social, Legal and Ethical Issues“ [online paper], The Royal Academy of Engineering, London, 2009, URL: http://www.raeng.org.uk/societygov/engineeringethics/pdf/Autonomous_Systems_Report_09.pdf [cited 29 June 2012].

12 Wooldridge, M. J. and Jennings, N. R., „Agent Theories, Architectures, and Languages: A Survey," in *Intelligent Agents III*, edited by J. P. Müller, Vol. 1193 of Lecture Notes in Computer Science, *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Heidelberg, 1997.