Towards Certification of Autonomous Unmanned Aircraft Using Formal Model Checking and Simulation

Matt Webster[*] and Neil Cameron[*]
Virtual Engineering Centre, Daresbury Laboratory, Warrington, UK

Mike Jump[†]
School of Engineering, University of Liverpool, Liverpool, UK

Michael Fisher[‡]
Department of Computer Science, University of Liverpool, Liverpool, UK

Unmanned aircraft are expected to increase in use in civil applications over the coming years, particularly for the so-called dull, dirty and dangerous missions. Unmanned aircraft will undoubtedly require some form of autonomy in order to ensure safe operations: communications failure could render a completely human-piloted unmanned aircraft dangerous to other airspace users. In order to be used for civil applications, unmanned aircraft must gain government regulatory approval in a process known as *certification*. This paper presents an approach to gathering evidence for certification of autonomous unmanned aircraft based on formal methods (in particular formal model checking) and flight simulation. In particular, rational agent-based autonomous systems are examined. Rational agents for unmanned aircraft can be model checked using implicit models of the aircraft's physical environment specified in terms of the different sensor inputs the autonomous system may receive. However this presents difficulties when trying to model check the agents relative to physical quantities such as those found in regulatory documents like the CAA Air Navigation Order. It is shown how this can be remedied using an explicit physical model of the environment within the model checker, and how this explicit physical model can itself be verified through comparison with flight simulations. To conclude, an overview of related and future work is given.

Nomenclature

LTL	Linear-time Temporal Logic	$^\circ$	Degrees of arc
$+!_a g$	Agent adds the desire to achieve g	AIL	Agent Infrastructure Layer
$+!_p g$	Agent adds the desire to perform g	BDI	Belief–Desire–Intention
$+f$	Agent adds the belief that f is true	CAA	Civil Aviation Authority
$-f$	Agent deletes the belief that f is true	COTS	Commercial Off-The-Shelf
$\Box p$	LTL: At all points in the future p is true	FAA	Federal Aviation Administration
$\Diamond p$	LTL: At some point in the future p is true	Hz	Hertz
$\neg p$	LTL: not p (i.e., p is not true)	JPF	Java PathFinder
$\top$	LTL: Logical truth	km	Kilometres
$p \mathsf{U} q$	LTL: p is true until q is true	PRISM	Probabilistic Symbolic Model Checker
$p \Rightarrow q$	LTL: p implies q	PROMELA	Process Meta-Language
$p \wedge q$	LTL: both p and q are true	SPIN	Simple PROMELA Interpreter
$\mathsf{B_a} p$	Agent a believes p	UAS	Unmanned Aircraft System
$\mathsf{D_a} p$	Agent a desires p	VEC	Virtual Engineering Centre
$\mathsf{I_a} p$	Agent a intends to do p	VESL	Virtual Engineering Simulation Lab.

[*]Postdoctoral Research Associate, Virtual Engineering Centre, Daresbury Laboratory, Warrington, Cheshire, WA4 4AD, UK.
[†]Lecturer, School of Engineering, University of Liverpool, Liverpool, Merseyside, L69 3GH, UK.
[‡]Professor, Department of Computer Science, University of Liverpool, Liverpool, Merseyside, L69 3BX, UK.

I. Introduction

It is anticipated that the use of unmanned aircraft within civil airspace will increase over the coming years.[1,2] To be economically viable, unmanned aircraft will have to be operated in non-segregated airspace, i.e., airspace that is also available to other users. To that end, unmanned aircraft will be subject to the same regulatory processes as manned aircraft and will hence require certification. During the certification process, evidence that a new aircraft type meets the regulations pertinent to its intended use is presented to a regulatory body (such as the Federal Aviation Administration (FAA) in the USA or the Civil Aviation Authority (CAA) in the UK). Once the regulator is satisfied that the regulations have indeed been complied with, a Type Certificate will be issued for the new aircraft type, thus permitting its use within that country's civil airspace.[1,3]

The certification process for manned aircraft is well-understood as the current regulations have evolved over 100+ years of manned aviation. However, unmanned aircraft present a significant technological and regulatory challenge. While the elements of an unmanned aircraft that would be present on a manned aircraft can be certificated under existing regulations, the autonomous aspects of unmanned aircraft must also be certificated, but using regulations that are still to be developed (and with far less experience than has been available for manned aviation). Nevertheless, any unmanned aircraft will need to be shown to be no less safe than a manned aircraft, equivalent to manned aircraft in its behaviour, and transparent to other airspace users and existing infrastructure (e.g., air traffic control).[4]

The potential civilian uses of autonomous unmanned aircraft are manifold: remote sensing, disaster response, surveillance, search-and-rescue, to name a few. While the military use of unmanned aircraft is increasing exponentially, the uptake in the civil sector is much slower.[1] One possible reason for this is that the potential manufacturers of autonomous unmanned aircraft and the regulatory bodies tasked with their certification are currently faced with a predicament. Manufacturers would like a set of regulations before investing in the development and manufacture of autonomous unmanned aircraft. However, regulators are hesitant to produce regulations when the full implications of the use of autonomous aircraft in civil airspace are unknown, as there are currently no such aircraft in routine use through unsegregated airspace. However, it is our view that *virtual prototypes*[5] of autonomous unmanned aircraft operating within realistic simulations of civil airspace may provide a solution to this problem as they can be tested as a real-life prototype would be prior to certification. Additionally, by taking place within a virtual environment these tests would be possible at a fraction of the costs implicit in real-world testing.[6]

It has been shown how formal model checking tools can be used to demonstrate human equivalence of autonomous unmanned aircraft systems through verifying autonomous systems relative to small subsets of the Rules of the Air, e.g., air traffic control clearance rules or rules corresponding to emergency avoidance scenarios, and how this might be applied to the problem of certification.[7] These Rules were checked using a model of the autonomous system's environment based on the information sent directly from sensors to the autonomous system. This environment model was implicit, i.e., the physical world was not modelled exactly, but rather in terms of the how the sensor systems would perceive that physical world. However, some Rules are more difficult to check in this way. For instance, certain Rules concern abstract quantities, e.g., one Rule concerns maintaining a minimum separation of 500 feet from all other aircraft.[8] In order to model check this Rule, a model of an environment would need to be constructed in which complex concepts such as distance, time and separation are present. By using such an explicit (or, at least, *more* explicit) environment model, quantitative Rules of the Air could be verified formally using a model checker. Additionally, by integrating an already formally-verified autonomous unmanned aircraft within a distributed, networked, synthetic environment of the type described by Cameron et al.,[9] the accuracy of the more explicit environment model can be verified.

Both of these approaches are investigated in this paper, and the way in which they might be used to gather evidence towards certification of unmanned aircraft is summarised in Figure 1. Agent programs are compiled, verified formally using model checking and integrated within a virtual prototype autonomous unmanned aircraft. Results of multiple simulation runs can be analysed and the information gained can be used to refine (i) the agent program, and (ii) the environment model used during formal model checking.

In Section II the concept of a rational agent is introduced, and it is shown how an autonomous agent-based control system for an unmanned aircraft was developed. In Section III model checking is introduced and it is shown how it can be used for formal verification of agent systems. In Section IV the development of a higher fidelity environment model for agent model checking can be constructed, based on physical models of the civil airspace environment. This enables quantitative requirements (e.g., certain Rules of the Air) to be formally verified. It is verified formally that an agent-based control system for an autonomous unmanned aircraft will always maintain at least 500 feet from an intruder aircraft. In Section V it is shown how the environment model can also be refined and made more accurate through verification using flight simulation software. Finally, in Section VI it is described how the combination of formal model checking and simulation can be used to generate evidence towards certification of unmanned aircraft.

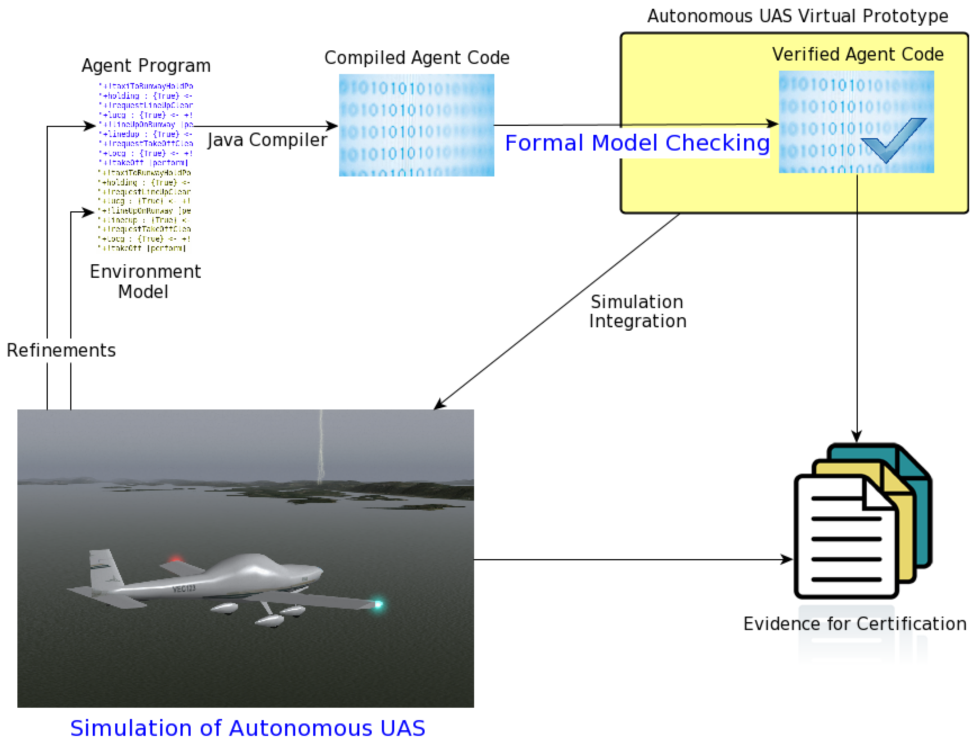


Figure 1. Using formal model checking and simulation to gather evidence for certification of autonomous unmanned aircraft.

Comparisons are made with similar work in the literature and directions for future research are given.

II. Agents and Autonomous Systems

An agent is a computer system situated within an environment that is capable of autonomous action within that environment towards its objectives. Agent systems have been used for numerous applications in which autonomous behaviours are desirable, including distributed sensing, electronic commerce and social simulation.[10]

In the work described in this paper, agent systems are used to manage and control the safe flight of autonomous unmanned aircraft operating in a virtual civil airspace environment. The agent architecture used in this paper is based on the Belief–Desire–Intention (BDI) model of *rational* agent reasoning, in which the agent has beliefs about its environment, together with desires (things which it would like to become true within its environment) and intentions (the particular deeds the agent deems necessary to achieve its desires based on its beliefs). It is important to point out that the BDI architecture is metaphorical; it is not suggested that agent systems have beliefs, desires or intentions in the full human cognitive sense. Rather, beliefs, desires and intentions are used as a basis for capturing autonomous behaviours. People tend to think in terms of beliefs, desires and intentions, and so this is a natural and expressive way to encode autonomous behaviours. Furthermore, use of the BDI architecture results in rational agents whose behaviour can be explained in terms of rationality and reasoning.

Gwendolen,[11] a BDI agent programming language, is the implementation language used to program abstract behaviours for an autonomous unmanned aircraft control system. Gwendolen agents are composed of a number of plans, each of which has the form:

```
event : guard <- deeds
```

Each plan begins with a triggering `event`, such as the addition of a belief or a message sent from another agent. Next is the `guard`, which consists of a set of propositions which may be true or false. Typically the guard is formed by a set of statements about the agent's beliefs, e.g., "the agent believes that its altitude is 10,000 feet". If a triggering event occurs and the statements in the guard are found to be true, then the `deeds` in the plan are enacted by the agent. The deeds can include actions within the agent's environment, sending messages to other agents, adding/removing beliefs in the agent's own belief database, and so on. The full formal syntax and semantics of the Gwendolen agent programming language is given by Dennis & Farwer.[11] However, a short guide to Gwendolen is given below:

- Events or deeds can be of the form: $+b$ or $-b$, representing the addition or deletion of a belief respectively.
- Guards can be about beliefs, e.g., $\mathsf{B}\ p$, meaning that "the agent believes p", or simply $\top$, meaning "true", which is always true.
- Goals (i.e., Desires) are represented as $!_a g$ meaning "a goal to achieve g," or $!_p g$ meaning "a goal to perform g". Events or deeds containing goals may have a $+$ or $-$ before the goal, representing that the agent has added/deleted that goal. One example would be $+!_p g$, meaning "the agent adds the goal to perform g".

A. Agents for Unmanned Aircraft

Rational agents are well-suited for use in autonomous unmanned aircraft, as they can provide the overall direction and control for a task or mission. Typically, beliefs are formed as a result of sensor inputs, which are distilled from continuous real-world data. For example, a sensor system might relay that, "there is an aircraft 541.2 feet away travelling on a heading of 220.8° at a speed of 161.3 knots." This could be stored precisely, e.g., $\mathit{aircraft}(541.2, 220.8, 161.3)$, or stored more abstractly, e.g., $\mathit{aircraft}(540, 220, 160)$ or even as $\mathit{safeSeparationFrom}(\mathit{aircraft})$. During the agent's reasoning process the agent may decide to take some action which is then forwarded on to actuators in control of the unmanned aircraft. For instance, an action $\mathit{turnToHeading}(0°)$ could map to a command to the heading control system to alter course until the heading is equal to 0°.

A generic architecture for the use of agents within unmanned aircraft (and autonomous systems in general) is given in Figure 2. Information flows into the control system and rational agent from the environment. Both components communicate, with the rational agent determining the overall control of the autonomous system, and the control system determining the outputs to actuators. This generic architecture can be used beyond unmanned aircraft, e.g., in ground vehicles, spacecraft, marine vehicles, robotics, etc.[12]

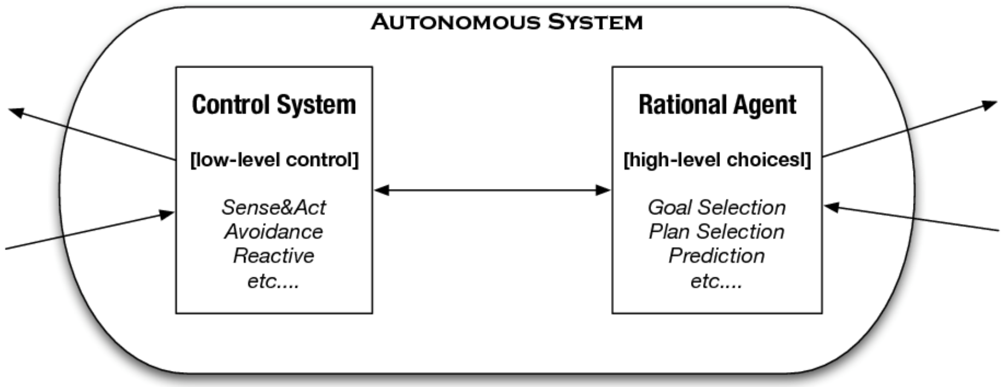


Figure 2. A generic architecture for rational agent-based autonomous systems.

A virtual prototype unmanned aircraft was developed using this architecture. The rational agent in overall control, known as the *Executive*, was written in Gwendolen and connected via network socket interfaces to the flight control system. The Executive is capable of controlling the autonomous flight of a virtual prototype unmanned aircraft through simulated civil airspace. The Executive is composed of a set of plans towards the successful completion of its missions. Missions are, in general, divided up into a number of flight phases representing the behaviour of the vehicle at different points of its mission:

1. Wait at the airport ramp.
2. Taxi to runway holding position.
3. Taxi to runway line-up position.
4. Take off.
5. Cruise.

6. Emergency avoid.
7. Aerodrome approach.
8. Land and taxi to a full stop.

The Executive's plans are designed to direct the autonomous flight of the unmanned aircraft through these flight phases towards successful mission completion. A subset of the Executive's plans is shown in Figure 3. The first plan says that when the agent has a new goal (i.e., desire), "*initiate*", it will form a goal to complete its mission. The second plan says that if the agent has been notified by the autopilot that the current flight phase is complete, and if it believes it is waiting at the ramp prior to take-off, then it will form intentions to send the new "*taxi*1" flight phase to the autopilot, delete the belief that the vehicle is waiting at the ramp, add the belief that the current flight phase is "*taxi*1" and delete the belief that the flight phase is complete. Likewise, the third plan moves the aircraft's flight phase from "*taxi*1" to "*taxi*2". The fourth and fifth plans deal with alerts concerning an intruder aircraft: if the alert is on, the agent goes into "*emergencyAvoid*" flight phase, and if the alert is off, the agent goes back into the flight phase it was in before the alert occurred. This particular version of the Executive agent consists of 23 such plans.

```
+!_p initiate : ⊤ <-
    +!_a completeMission

+flightPhaseComplete : B flightPhase(waitingAtRamp) <-
    +.lock, send2fl(flightPhase(taxi1)), −flightPhase(waitingAtRamp), +flightPhase(taxi1), −flightPhaseComplete, −.lock

+flightPhaseComplete : BflightPhase(taxi1) <-
    +.lock, send2fl(flightPhase(taxi2)), −flightPhase(taxi1), +flightPhase(taxi2), −flightPhaseComplete, −.lock

                ...

+intruderHeadOnAlertOn : B flightPhase(X) <-
    +.lock, −intruderHeadOnAlertOn, +flightPhase(emergencyAvoid), +store(X), send2fl(emergencyAvoidOn), −.lock

+intruderHeadOnAlertOff : B flightPhase(emergencyAvoid), B store(X)) <-
    +.lock, −intruderHeadOnAlertOff, +flightPhase(X), −store(X), send2fl(emergencyAvoidOff), −.lock
```

Figure 3. A subset of the Executive's plans, written in Gwendolen.

III. Formal Model Checking

Formal model checking, also known simply as model checking, is an approach to verification of computer systems based on an exhaustive exploration of the entire state space of a program or model of a program.[13] The practical uses of model checking began in the early 1980s, with the theoretical basis for model checking being developed by various researchers around that time.[14] Originally developed for the verification of concurrent systems and communications protocols, model checking has since been used to formally verify a variety of safety- and mission-critical software and hardware (e.g.,[15,16,17]). The exhaustive nature of model checking provides a high level of confidence that the system under analysis satisfies the properties being verified. The mathematical basis of model checking distinguishes model checking, as well as other approaches based on formal methods, as offering *formal verification*.

Our approach is based on the Agent JPF model checker, itself based on the Java PathFinder (JPF)[18] model checker developed at NASA Ames Research Center. JPF was developed to verify the behaviour of Java programs. JPF examines executable object code (rather than a model of it, or its source code) in contrast to other model checkers such as SPIN[13] or PRISM.[19] It is for this reason that JPF is more accurately termed a *program checker*. (However, "model checker" will be used for the remainder of this paper to avoid confusion.) Agent JPF works by verifying the behaviour of a single- or multi-agent program written in Java. Agent JPF is complemented by the Agent Infrastructure Layer (AIL), an application programming interface for the rapid creation of BDI agent programming languages in Java. Both Agent JPF and the AIL were originally developed as part of the Model Checking Agent Programming Languages (MCAPL) project at the Universities of Liverpool and Durham,[20] and are still under development at the University of Liverpool.[21]

In order to use Agent JPF to model check an agent system, the agent system can be written using the Gwendolen agent programming language. Then, the agent program(s) are compiled using Java and the AIL into Java executable code. Next, the property to be verified (typically derived from a requirement of the system) is formalised using linear-time temporal logic (LTL),[22] which allows concepts about time to be encoded in a formal mathematical framework. Finally, the Agent JPF model checker is run. Agent JPF takes as its inputs the program to be model checked and the property to be verified and returns a true/false answer based on whether the program satisfies the property. If the program does not satisfy the property, the error state can be printed as well as the states leading up to the error state in order to assist debugging.

LTL properties are essential for model checking using Agent JPF. For this reason the following short guide to a subset of LTL is given:

- $\Box p$, read as "from now on, it is always the case that p is true".
- $\Box\neg p$, read as "from now on, it is always the case that p is false".
- $\Diamond p$, read as "from now on, at some point p will be true".
- $\Diamond(p \wedge q)$, read as "from now on, at some point both p and q will be true".
- $p\ U\ q$, read as "from now on, p will be true until q is true".

(Note that these operators have precise formal definitions (see, e.g.,[23]) as well as the informal definitions given here.)

Rational agents can be model checked through extensions to the logic described above. For example:

- $\Box \mathsf{B}_{\mathsf{a}} p$, read as "from now on, it is always the case that agent a believes p is true".
- $\Box \mathsf{D}_{\mathsf{a}} p$, read as "from now on, it is always the case that agent a desires p".
- $\mathsf{D}_{\mathsf{a}} p\ \mathsf{U}\ \mathsf{I}_{\mathsf{a}} q$, read as "from now on, agent a will desire p until it intends to do q".

Note that if only one agent is being considered, it is sufficient to say simply "$\mathsf{B}p$" rather than "$\mathsf{B}_{\mathsf{a}}p$", for example.

In this paper the properties tested are all safety properties, which take the form $\Box\neg bad$, where *bad* is some undesirable condition. Other kinds of properties that could be tested include liveness properties (e.g., $\Diamond good$ — at some point in the future something good will happen) or fairness properties (e.g., $\Box\Diamond send \Rightarrow \Box\Diamond receive$ — if a message is sent infinitely often, then it is received infinitely often as well).

IV. Higher Fidelity Model Checking

Webster et al.[7] describe how rational agents for unmanned aircraft can be model checked using a model of the agent's environment consisting of three components:

- A sensor unit. This represents a detect-and-avoid sensor which alerts the agent whenever an object is approaching head-on.
- A navigation manager. This represents a navigation subsystem which alerts the agent whenever the vehicle must correct its course in order to reach a pre-planned destination.
- An air traffic control transceiver. This communicates with aerodrome air traffic control and notifies the agent if different clearances have been given or denied, e.g., clearance to taxi or take-off.

For example, the sensor unit may send a message, "aircraftApproachingHeadOn" to alert the agent that this is the case, and may send a message "aircraftAverted" to alert the agent that the aircraft is no longer approaching head on. The navigation manager and air traffic control models behave in a similar way. It is clear that the model of the environment is explicit relative to the messages being sent between components, but *implicit* relative to physical reality. For example, the position of the intruder aircraft is not modelled explicitly; nowhere in the model is this data stored. Rather, the effect of an intruder aircraft is modelled; in this case, an alert from a sensor unit saying that there is an intruder aircraft. Likewise, the navigation of the vehicle through airspace is not modelled explicitly, nor are the air traffic conditions at the aerodrome under air traffic control.

This kind of environment modelling is based on an understanding of the design of the autonomous system in which the agent is based. All that is needed is an understanding of the protocol by which the agent communicates with other (electronic) components on the unmanned aircraft and how those components reflect and report the physical world to the agent. Indeed, this kind of protocol-based model has been used with considerable success when model checking mission-critical software, (see, e.g.,[15]).

However, in the case of agent model checking for unmanned aircraft, this kind of implicit environment modelling presents difficulties. Frequently it is desirable to verify the behaviour of an agent-based autonomous unmanned aircraft using the Rules of the Air, many of which include physical quantities, e.g.:

- **The 500 feet rule**: "…an aircraft shall not be flown closer than 500 feet to any person, vessel, vehicle or structure." (Section 2.3.5.3(b))[8]
- **The 1,000 feet rule**: "…an aircraft flying over a congested area of a city, town or settlement shall not fly below a height of 1,000 feet above the highest fixed obstacle within a horizontal radius of 600 metres of the aircraft." (Section 2.3.5.3(c))[8]
- **Aerodrome Ground Visibility**: "…an aircraft which is unable to communicate by radio with an air traffic control unit at the aerodrome of destination shall not begin a flight to the aerodrome if… the weather reports and forecasts which it is reasonably practicable for the commander of the aircraft to obtain indicate that it will arrive at that aerodrome when the ground visibility is less than 10 km or the cloud ceiling is less than 1,500 feet." (Section 2.4.7.3(b))[8]

In implicit models, these physical quantities are not present by definition. For example, a sensor system may report that an intruder aircraft is "approaching", but that aircraft's physical attributes such as heading, speed, altitude, etc., are not modelled explicitly. In order to formally verify properties which contain physical quantities a more explicit environment model is required. The use of more realistic environment models within model checking has a number of advantages over simulation and other model checking techniques such as those described by Webster et al.:[7]

1. Physical quantities can be modelled, meaning that requirements based on physical quantities (such as those given above) can be verified formally.
2. Flight simulation software can be used to verify the assumptions made in the environment model. In this case simulation is not being used directly to generate evidence for certification but is helping to verify the assumptions made in the model being used for certification.

A. Developing a Higher Fidelity Environment Model for Model Checking

In order to develop a higher fidelity environment model for model checking, a detect-and-avoid scenario that had previously been developed in a flight simulation was analysed. In the scenario, an unmanned aircraft is flying straight and level through simulated UK civil airspace in the "cruise" flight phase, and encounters a single intruder aircraft approaching approximately head-on. A sensor system registers the intruder, which informs the Executive agent, which decides to implement an "emergency avoid" manoeuvre consisting of a roll to the right (in accordance with the Rules of the Air). When the sensor system registers that the intruder aircraft has been successfully avoided, it informs the Executive agent which decides to cancel the Emergency Avoid manoeuvre and return to the "cruise" flight phase. When the flight control system receives this command, it executes a turn to the left in order to resume its original course. The manoeuvre is illustrated in Figure 4.

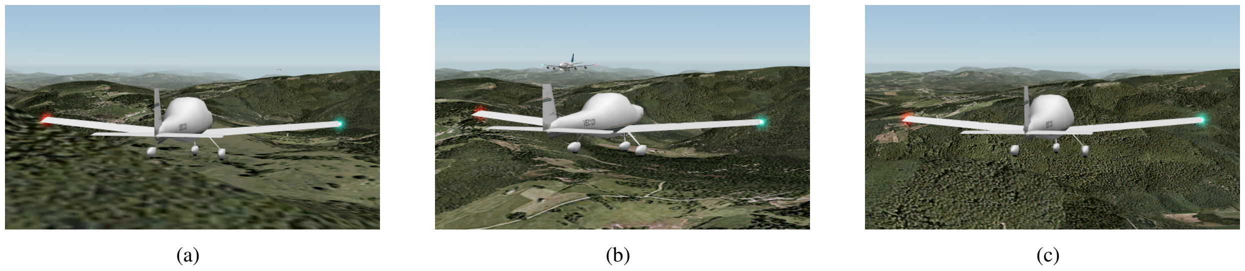

(a) (b) (c)

Figure 4. A simulated detect-and-avoid manoeuvre in the VESL flight simulator. Image (a) is prior to the manoeuvre, image (b) is after the first turn and image (c) is after the second turn.

The flight simulation environment used is the VEC's Virtual Engineering Simulation Laboratory (VESL) described by Cameron et al.[9] VESL uses Advanced Rotorcraft Technology's FLIGHTLAB® software[24] for high fidelity flight dynamics simulation. FLIGHTLAB is connected via custom software and hardware interfaces to the Executive agent described in Section A. The agent is kept informed of changes in its environment through simulated sensors, one of which is a detect-and-avoid sensor system. Once the Executive agent has decided on a course of action, it sends a message to the flight control system (also simulated in FLIGHTLAB) which determines the control inputs required to execute the Executive agent's command. For example, the Executive agent may decide to alter course to a heading of 210°, which it will send in a message to the flight control system. The flight control system will then calculate and implement the exact combination of simulated actuator inputs required to cause the vehicle to turn onto the desired heading. A schematic illustration of the VESL's architecture is given in Figure 5.

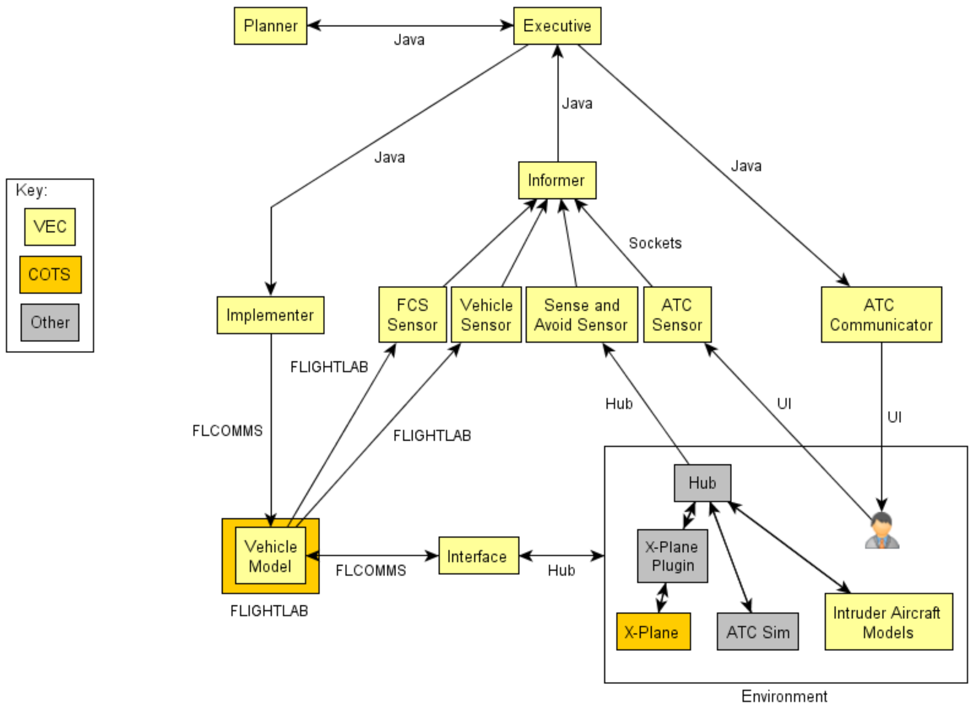


Figure 5. Software architecture of the Virtual Engineering Simulation Laboratory (VESL) at the VEC. Different coloured boxes represent the origin of the software systems. Arrows represent information flow between software systems, and labels on the arrows show the software/protocol used. More information on the VESL is given by Cameron et al.[9]

As the aim was to create a higher fidelity environment model for the purposes of model-checking the Executive agent, the detect-and-avoid manoeuvre from the flight simulation was recorded and analysed. Since the manoeuvre consists of two turns, one to the right and one to the left, the headings of the unmanned aircraft in the simulation were noted at three points: (i) prior to the manoeuvre, (ii) after the first turn; and (iii) after the second turn. Furthermore, the duration of time between (ii) and (iii) was noted. The sensor model used in the detect-and-avoid simulation was a simple proximity sensor, set to detect any intruder aircraft within 20,000 feet (6,096 metres)[a] This proximity sensor was modelled in a similar way within the model checking environment model. The intruder aircraft in the simulation was set to approach the unmanned aircraft head-on, and this was also modelled in a similar way.

The Executive agent described in Figure 3 was reduced to a core set of plans necessary for detect-and-avoid functionality. The Executive agent's execution environment (written using the AIL) was adapted to provide a simple (yet explicit) physical model of the unmanned aircraft's flight through airspace, based on heading, speed, latitude and longitude. (The altitude was assumed to be constant for the unmanned aircraft and the intruder, as the altitudes of the unmanned aircraft and the intruder were constant for the duration of the scenario in the VESL flight simulator.)

When the Executive agent is executed within the higher fidelity environment model, the agent directs the modelled aircraft through the simulated airspace. The intruder aircraft flies in the opposite direction to the unmanned aircraft on a head-on collision course. At a distance of 20,000 feet a sensor system registers the presence of the in-

[a]Note that this proximity sensor was a simple "placeholder" model designed to work in the case of a single uncooperative intruder aircraft, and did not monitor whether the intruder was on a collision course. Of course, the proximity sensor model works in the scenario being examined here. Higher fidelity detect-and-avoid sensor models are being developed to tackle more elaborate scenarios.

truder aircraft and informs the Executive agent. The Executive then decides to engage the "emergency avoid" flight phase. The manoeuvre described above is conducted until the sensor system detects that the intruder aircraft has gone out of range (20,000 feet) and informs the Executive agent. At this point the Executive resumes the "cruise" flight phase. Illustrations of the detect-and-avoid scenario as modelled in the higher fidelity environment model are given in Figure 6.

B. Model Checking Using the Higher Fidelity Environment Model

Using the higher fidelity environment model it was possible to formally verify a Rule of the Air based on physical quantities used in the model:

- **The 500 feet rule**: "…an aircraft shall not be flown closer than 500 feet to any person, vessel, vehicle or structure." (Section 2.3.5)[8]

Using the environment model described above, the Executive agent was model checked with respect to the following LTL property corresponding to the Rule above:

$$\Box \neg \mathsf{B}(\text{Executive}, \text{separationUnsafe})$$

Here "separationUnsafe" is a statement that is true if and only if the distance between the unmanned aircraft and the intruder aircraft is less than 500 feet. The belief "separationUnsafe" will be added to the Executive's belief database only in the event that the distance is less than 500 feet. Therefore if this property holds (i.e., is verified as true by the model checker) then we know that the Executive agent follows this Rule of the Air.

In this case, the model checker checks only the case where the sensor can detect aircraft at a range of up to 20,000 feet, i.e., it checks a single run of the model in which the unmanned aircraft performs the detect-and-avoid manoeuvre. Note that model checking the program in this case is different from simply executing the program once. In execution, the Java runtime environment executes the Java bytecode, performing whichever outputs are specified in the program. However, in model checking the Agent JPF model checker executes the Java bytecode instead, and examines every step in the execution path to check whether the property being checked is true. If at any point the property evaluates to false the model checker will stop execution of the program, along with output about the error-causing state and how that state came to be (i.e., an error trace detailing the states leading up to the error state). It is possible to arrive at the conclusion that model checking in this case is equivalent to testing; however the fact that the state is being analysed at every point together with error trace data according to properties *specified formally* means that model checking offers more than software testing, even in this simple case.

It is possible to use the model checker in a more complex way to analyse a range of different scenarios automatically, and therefore give a high level of assurance that the property being checked holds in all relevant cases. The example above was adapted to vary the sensor detection range r between 16,700 feet[b] and 20,000 feet in 3.28 feet (= 1 metre) intervals in order to determine the effects of reduced sensor range on the detect-and-avoid manoeuvre. This variable was built into the environment model using the Verify.getInt(x,y) method in the Agent JPF model checker, which allows a variable to be altered within a range at model checking run-time[c]. The relevant line of Java was:

```
sensorRange = (double) Verify.getInt(5096, 6096);        // 6,096m = 20,000 feet
```

The model checker was used to verify formally that the property above was satisfied for all sensor ranges from 16,700 feet to 20,000 feet. Note that the model checker can also be used to output flight parameters at run-time. (E.g., the minimum separation distance between the unmanned aircraft and the intruder aircraft was found to be 944 feet when the sensor range was 16,700 feet, and 1,129 feet when the sensor range was 20,000 feet.)

V. Verifying Model Checking Results using Simulation

In the previous section the development of a higher fidelity environment model for model checking was described. In this section it is described how these kinds of model can then be verified and re-engineered based on comparisons with flight simulation models.

[b]The reason 16,700 feet was chosen is that this is approximately 1000 metres less than the maximum sensor range of 20,000 feet. The use of two different units for distance was an unavoidable result of the use of more than one simulation environment in this work.

[c]In the case of execution outside the model checker, i.e. in the Java runtime environment, this method returns a pseudorandom number in the specified range.

After execution of the program containing the Executive agent and the higher fidelity environment model, it was found that the minimum separation distance from the unmanned aircraft to the intruder aircraft was 1,129 feet when the detect-and-avoid sensor range was set to 20,000 feet. Analysis of the flight simulation of the same scenario showed that the minimum separation distance between the unmanned aircraft and the intruder aircraft was 820 feet. Clearly there is a difference between the two models of 309 feet. After comparing the outputs of the two models it was found that in the higher fidelity environment model the turns in the Emergency Avoid manoeuvre were based only on changes in the heading of the modelled vehicle, e.g., the vehicle turns immediately from a heading of 0° to 10.6° in a discontinuous way (see Figure 6). It was hypothesised that the reason for the 309 feet disparity was due, in part, to the relatively unrealistic way in which the turns were modelled compared to the flight simulation, in which the turns were gradual and continuous. So, in order to improve the accuracy of the higher fidelity environment model the output of the simulator model was analysed to discover the turn rate of the unmanned aircraft during its two turns. It was found that the turn rate in both turns was 0.83° per second. The higher fidelity environment model was then modified to incorporate this turn rate. Running the simulation again revealed that the minimum separation distance of the unmanned aircraft and the intruder had now reduced to 929 feet — a 109 feet difference, much less than the original 309 feet difference. The effect of the revised higher fidelity environment model on the unmanned aircraft's flight path can be seen in Figure 7.

After improving the accuracy of the environment model, the Executive agent was model checked again to ensure that the 500 feet rule requirement was still satisfied for all sensor ranges between 16,700 feet and 20,000 feet.

VI. Summary & Conclusions

In this paper an approach to generating evidence towards certification of autonomous unmanned aircraft has been described. The approach can be summarised thus:

1. Generation of a virtual prototype agent-based control system for an autonomous unmanned aircraft together with a low-fidelity implicit model of the environment based on sensor inputs. (This step was described by Webster et al.[7])

2. Integration of the agent-based control system within a virtual prototype unmanned aircraft within a flight simulator (VESL) in order to execute a high fidelity detect-and-avoid scenario simulation. (This step was described by Cameron et al.[9])

3. Discovery of the flight profile of the autonomous unmanned aircraft detect-and-avoid manoeuvre based on analysis of data from the high fidelity flight dynamics simulation software within the flight simulator, and discovery of the minimum separation distance in this scenario.

4. Generation of a higher fidelity environment model for model checking based on an explicit (yet simple) physical model of the unmanned aircraft detect-and-avoid scenario.

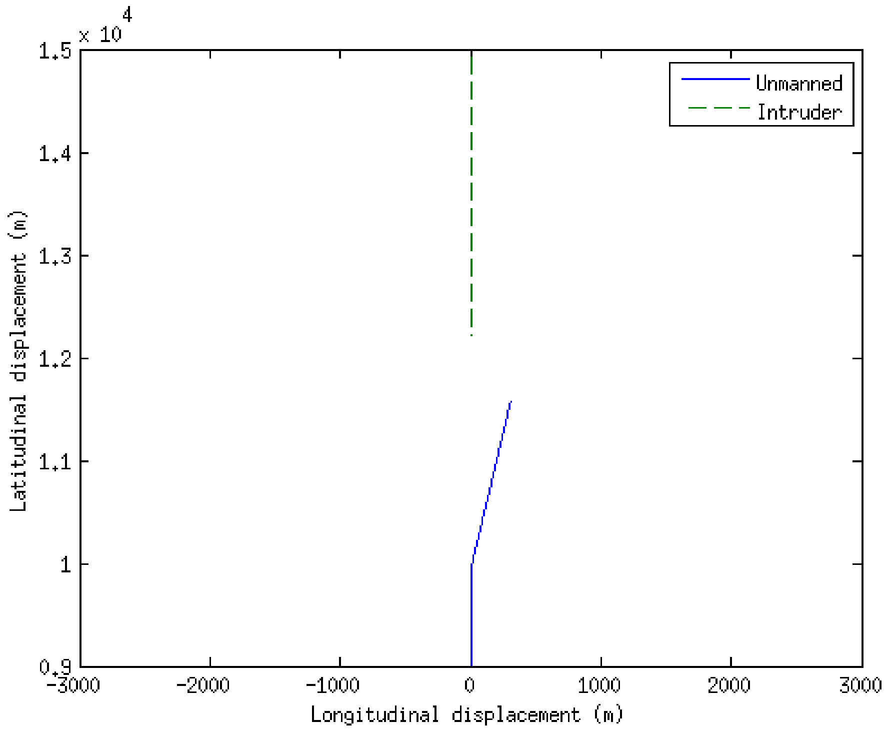
5. Model checking the Executive agent based on this higher fidelity environment model to verify formally that a quantitative requirement (based on one of the Rules of the Air) was met. Introduction of an modelled error into the detect-and-avoid sensor (to simulate reduced sensor range) in order to verify formally that the virtual prototype agent-based control system would still satisfy the quantitative requirement.

6. Discovery of the minimum separation distance between the unmanned aircraft and the intruder aircraft within the higher fidelity environment model used for model checking by executing the model directly (i.e., outside the confines of the model checker). Identification that the minimum separation distance was larger than the same distance measured in the flight simulation environment (by a margin of 309 feet).

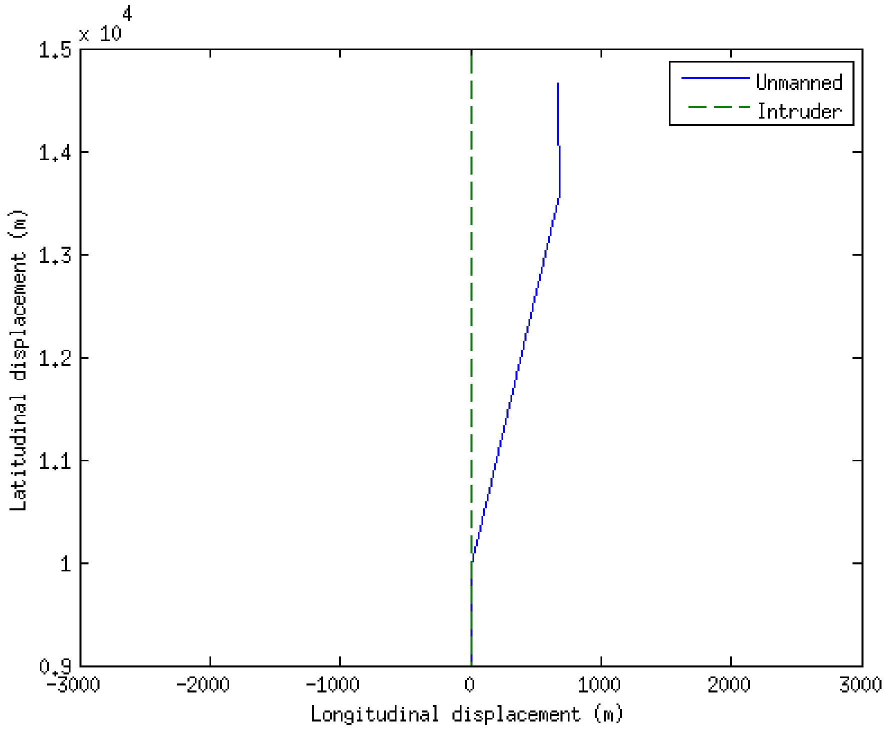
7. Revision of the higher fidelity environment model used for model checking to incorporate a more realistic turn profile for the unmanned aircraft during its detect-and-avoid manoeuvre.

8. Discovery of a new minimum separation distance as a result of the revised higher fidelity environment model, showing that the minimum separation distance in the model is now only 109 feet more than in the flight simulation of the same detect-and-avoid scenario.

9. Model checking the Executive agent based on this revised higher fidelity environment model to formally verify that the quantitative requirement is still met by the agent-based control system.

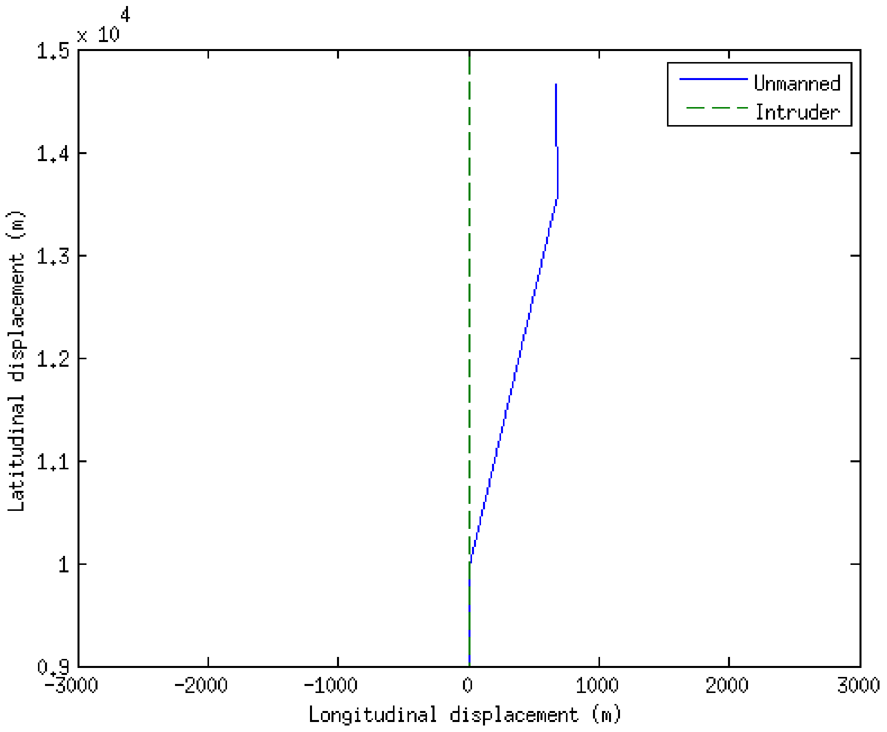


(a)

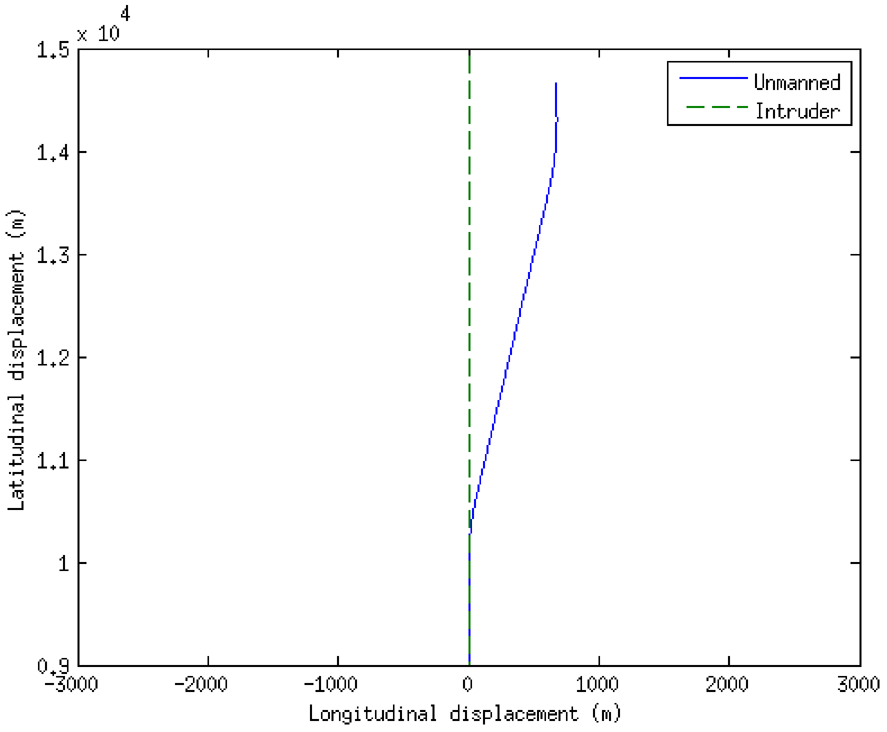


(b)

Figure 6. A simulated detect-and-avoid manoeuvre, as captured by the higher fidelity environment model. Diagrams (a) and (b) show the manoeuvre at the same points as in Figures 4(b) and 4(c) respectively. The unmanned aircraft is shown in blue and starts its run at the bottom of the figure, and the intruder aircraft is shown in red and starts from the top of the figure. Note that the intruder aircraft is uncooperative and does not alter its course in response to the presence of the unmanned aircraft.



(a)



(b)

Figure 7. Differences between the higher fidelity environment model and the revised higher fidelity environment model after including a turn rate in the detect-and-avoid manoeuvre. Diagram (a) shows the manoeuvre in the higher fidelity environment model, and diagram (b) shows the same manoeuvre in the revised higher fidelity environment model. Note that in diagram (b) the turns are smoother than in diagram (a).

This approach follows on from previous work by Webster et al.[7] on the use of agent model checking to formally verify selected Rules of the Air for autonomous control systems for unmanned aircraft, and extends it to cover requirements based on physical quantities. Furthermore, it is shown how the flight simulation facilities developed by Cameron et al.[9] can be used to verify the environment models required to model check those requirements.

We have shown how it is possible to model check requirements based on physical quantities (such as the Rules of the Air) for agent-based autonomous unmanned aircraft. This was done through the development of an environment model (called the "higher fidelity environment model") based on an abstraction of a detect-and-avoid flight simulation scenario. The environment model was used to provide more realistic inputs to the agent controlling the unmanned aircraft and therefore provide a higher level of assurance that the agent was implemented correctly. Furthermore, the environment model was refined after comparing its output to the flight simulation and discovering that the minimum separation distance of the unmanned aircraft and the intruder aircraft was significantly greater in the environment model. Overall this demonstrates how formal model checking and flight simulation can be used together to provide higher levels of assurance of the safety of autonomous unmanned aircraft. The outputs of the model checker, combined with the higher fidelity environment models, as well as model comparison and refinement based on flight simulation, could be presented as evidence to a regulatory authority such as the CAA in the UK or the FAA in the USA that a rational agent-based autonomous control system for an unmanned aircraft would be safe for use in civil airspace, and that an aircraft using such a system could be certificated. Furthermore, the approach described here is generic and could be applied to other manoeuvres and scenarios for unmanned aircraft beyond the simple detect-and-avoid scenario we have used.

This paper is part of a larger project to investigate means of gathering evidence (using formal methods and simulation) to present to national regulatory authorities (such as the CAA) to support certification of autonomous unmanned aircraft. Unmanned aircraft are likely to require the use of autonomous systems. Even where unmanned aircraft are remotely piloted, autonomous systems will be required in the case where communications with a ground-based human operator have failed. The authors anticipate that over time the advantages of autonomous systems for unmanned aircraft — such as verifiability, reliability and cost — will catalyse their adoption in civil unmanned aviation and beyond.

A. Comparisons with Related Work

Formal methods, of which model checking is one, have also been used to analyse and verify unmanned aircraft. Formalisms used include SPARK Ada,[25] Event-B[26] and Kripke models.[27,28] A brief overview of a number of uses of formal methods (such as model checking) for formal verification of unmanned aircraft is given by Webster et al.[7]

A similar approach to ours is given by Bass et al.,[29] who describe formal verification of collaboration between humans and agents in aerospace applications based on model checking, simulation and abstract modelling of hybrid systems. Their work is part of the *NextGen Authority and Autonomy* project on the next generation of air traffic control procedures involving the automated exchange of information between pilots, ground controllers and automated systems in aircraft and on the ground. The broad aim is to ensure that there is no potential for unexpected behaviour in the NextGen system (known as "Automation Surprise"). Bass et al. develop models of humans and automated systems using the agent paradigm, which are model checked using an SMT (Satisfiability Modulo Theories) solver. It is demonstrated how unexpected behaviour can be discovered using the model checker for a scenario involving the Airbus A320 automated speed protection functionality. The key similarity between the work of Bass et al. and the approach described in this paper is that simulation and model checking are used together to develop a level of assurance that aircraft computer systems will behave as expected. However, the level of agent modelling differs in the two approaches; Bass et al. use an abstract model of human and machine agency, whereas in this paper the agents verified are more detailed and concrete as they are embodied in executable agent programs.

Clarke et al.[30] present an approach to model checking programs with large numbers of states based on an abstraction of the program. In one case the authors are able to model check a program with over 10^{1300} states. Our approach is similar, except that we do not model check an abstraction of a program but rather the program itself. However, we do form an abstraction of the environment, as real world environments are likely to have an extremely large (if not infinite) number of states.

Platzer & Clarke[31] describe how formal verification can be applied to collision avoidance manoeuvres, in particular the "fully flyable tangential roundabout manoeuvre" (FTRM) in which two aircraft in danger of collision are able to safely manoeuvre away through the joint application of the manoeuvre. The authors use a proof assistant for non-linear hybrid systems to prove that in all cases the FTRM is collision-free. While we have used a similar example of a collision avoidance manoeuvre in this work, our work is intended to be indicative of an approach towards gathering evidence for certification of autonomous unmanned aircraft, and could be applied to other kinds of manoeuvre or scenario.

B. Future Work

The work described in this paper is a proof-of-concept of the use of higher fidelity environment models to model check quantitative requirements for autonomous systems. However, the approach can be developed and expanded in a number of ways. Firstly there are many more Rules of the Air which are based on physical quantities (a small sample is given in Section IV). One obvious extension of this work would be to develop higher fidelity environment models to enable these Rules of the Air to be model checked as well. Note that this approach is not confined to the Rules of the Air; these are used only as clear examples of quantitative requirements, and it is likely that there are many more quantitative requirements beyond the Rules of the Air.

It is worth noting that the approach described in this paper is not limited to unmanned aircraft. In fact, any autonomous system using the abstract architecture given in Figure 2 that is embodied in a physical environment could be analysed using this approach, including unmanned ground vehicles, robots for manufacture or assisted living, autonomous space vehicles, etc.[12]

This paper contains an example of how model checking and simulation might be used to gather evidence towards certification of an autonomous unmanned aircraft. However, the software tools used in this paper (particularly the Agent JPF model checker) would need to undergo a rigorous *tool qualification*[32] analysis before evidence generated by the tools could be accepted by a regulatory authority like the CAA. Model checking and other formal methods are increasingly seen as viable tools for generating evidence for certification, as can be seen in the increasing attention given to them in standards like DO-178C,[33,34] compared to its predecessor, DO-178B.[32] However this trend must be complemented by an open-minded approach by regulators and manufacturers in order to maximise the benefits of formal model checking and simulation in the certification of aircraft and other safety-critical systems.

Acknowledgment

This work is supported through the Virtual Engineering Centre (VEC), which is a University of Liverpool initiative in partnership with the Northwest Aerospace Alliance, the Science and Technology Facilities Council (Daresbury Laboratory), BAE Systems, Morson Projects and Airbus (UK). The VEC is funded by the Northwest Regional Development Agency (NWDA) and European Regional Development Fund (ERDF) to provide a focal point for Virtual Engineering research, education and skills development, best practice demonstration, and knowledge transfer to the aerospace sector. For more information see http://virtualengineeringcentre.com/.

References

[1]Weibel, R. E. and Hansman, R. J., "Safety Considerations for Operation of Unmanned Aerial Vehicles in the National Airspace System," Tech. Rep. ICAT-2005-1, MIT International Center for Air Transportation, 2005.

[2]Zaloga, S. J., Rockwell, D., and Finnegan, P., "World Unmanned Aerial Vehicle Systems: 2011 Market Profile and Forecast," Teal Group Corporation, 2011, http://tealgroup.com. Accessed 2012-05-28.

[3]Civil Aviation Authority, "CAP 553 BCAR Section A: Airworthiness Procedures where the CAA has Primary Responsibility for Type Approval of the Product," http://www.caa.co.uk/docs/33/CAP553.PDF, October 2011.

[4]Civil Aviation Authority, "CAP 722 Unmanned Aircraft System Operations in UK Airspace — Guidance," http://www.caa.co.uk/docs/33/CAP722.pdf, April 2010.

[5]Cooper, J., Abdelal, G., Cameron, N., Fisher, M., Forrest, J., Georgiou, G., Hon, K., Jump, M., Padfield, G., Robotham, A., Shao, F., Webster, M., and White, M., "Virtual Engineering Centre — Examples of Virtual Prototyping and Multidisciplinary Design Optimization," *Proceedings of the NATO RTO Workshop AVT-173 on Virtual Prototyping of Affordable Military Vehicles Using Advanced MDO, Sophia, Bulgaria*, May 2011.

[6]Jones, A., "UAS Virtual Certification," Royal Aeronautical Society Unmanned Aircraft Systems Annual Two Day Conference, October 2011, http://www.astraea.aero/downloads/RAeS%20Conference%202011/ASTRAEA_VC_overview_RAES_2011_V1.ppt.pdf.

[7]Webster, M., Fisher, M., Cameron, N., and Jump, M., "Formal Methods for the Certification of Autonomous Unmanned Aircraft Systems," *The 30th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2011)*, edited by F. Flammini, S. Bologna, and V. Vittorini, Vol. 6894 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 228–242.

[8]Civil Aviation Authority, "CAP 393 Air Navigation: The Order and the Regulations," http://www.caa.co.uk/docs/33/CAP393.pdf, April 2010.

[9]Cameron, N., Webster, M., Jump, M., and Fisher, M., "Certification of a Civil UAS: A Virtual Engineering Approach," *AIAA Modeling and Simulation Technologies Conference, Portland, Oregon, Aug. 8-11, 2011*, No. AIAA-2011-6664, 2011.

[10]Wooldridge, M., *An Introduction to Multiagent Systems*, John Wiley & Sons, 2002.

[11]Dennis, L. A. and Farwer, B., “Gwendolen: A BDI Language for Verifiable Agents,” *Logic and the Simulation of Interaction and Reasoning*, AISB’08 Workshop, 2008.

[12]Fisher, M., Dennis, L., and Webster, M., “Verifying Autonomous Systems,” under review.

[13]Holzmann, G., *The Spin Model Checker: Primer and Reference Manual*, AW, 2004.

[14]Clarke, E. M., “The Birth of Model Checking,” *25 Years of Model Checking:History, Achievements, Perspectives*, edited by O. Grumberg and H. Veith, Vol. 5000 of *Lecture Notes in Computer Science*, Springer, 2008.

[15]Havelund, K., Lowry, M., Park, S., Pecheur, C., Penix, J., Visser, W., and White, J. L., “Formal Analysis of the Remote Agent Before and After Flight,” *The Fifth NASA Langley Formal Methods Workshop, Virginia, USA*, 2000.

[16]“National Highway Traffic Safety Administration Toyota Unintended Acceleration Investigation - Appendix A,” NASA Engineering and Safety Center Technical Assessment Report, 2011.

[17]“On-the-Fly, LTL Model Checking with SPIN: Inspiring Applications,” `http://spinroot.com/spin/whatispin.html`, May 2012, Accessed 2012-05-24.

[18]Visser, W., Havelund, K., Brat, G. P., Park, S., and Lerda, F., “Model Checking Programs,” *Automated Software Engineering*, Vol. 10, No. 2, 2003, pp. 203–232.

[19]Kwiatkowska, M., Norman, G., and Parker, D., “PRISM: Probabilistic Symbolic Model Checker,” *Proc. 12th Int. Conf. Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS)*, Vol. 2324 of *LNCS*, Springer, 2002, pp. 200–204.

[20]Dennis, L. A., Fisher, M., Webster, M. P., and Bordini, R. H., “Model Checking Agent Programming Languages,” *Automated Software Engineering*, Vol. 19, No. 1, March 2012, pp. 5–63.

[21]“Model-Checking Agent Programming Languages,” `http://mcapl.sourceforge.net`.

[22]Fisher, M., *An Introduction to Practical Formal Methods Using Temporal Logic*, Wiley, 2011.

[23]Baier, C. and Katoen, J.-P., *Principles of Model Checking*, MIT Press, Cambridge, MA, USA, 2008, ISBN 978-0-262-02649-9.

[24]Advanced Rotorcraft Technology, Inc., “FLIGHTLAB,” `http://www.flightlab.com/flightlab.html`. Accessed 2012-05-28.

[25]Sward, R. E., “Proving Correctness of Unmanned Aerial Vehicle Cooperative Software,” *Proc. IEEE International Conference on Networking, Sensing and Control*, 2005.

[26]Chaudemar, J.-C., Bensana, E., and Seguin, C., “Model Based Safety Analysis for an Unmanned Aerial System,” *Proc. Dependable Robots in Human Environments (DRHE)*, 2010.

[27]Jeyaraman, S., Tsourdos, A., Zbikowski, R., and White, B., “Formal Techniques for the Modelling and Validation of a Co-operating UAV Team that uses Dubins Set for Path Planning,” *Proc. American Control Conference*, 2005.

[28]Sirigineedi, G., Tsourdos, A., Zbikowski, R., and White, B. A., “Modelling and Verification of Multiple UAV Mission Using SMV,” *Proc. FMA-09*, Vol. 20 of *EPTCS*, 2009.

[29]Bass, E. J., Feigh, K. M., Gunter, E., and Rushby, J., “Formal Modeling and Analysis for Interactive Hybrid Systems,” *Proceedings of the Fourth International Workshop on Formal Methods for Interactive Systems (FMIS 2011)*, edited by J. Bowen and S. Reeves, Electronic Communications of the EASST, 2011, ISSN 1863-2122.

[30]Clarke, E. M., Grumberg, O., and Long, D. E., “Model Checking and Abstraction,” *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 5, 1994, pp. 1512–1542.

[31]Platzer, A. and Clarke, E. M., “Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study,” *FM 2009: Formal Methods Second World Congress*, edited by A. Cavalcanti and D. R. Dams, Vol. 5850 of *Lecture Notes in Computer Science*, Springer, 2009.

[32]RTCA Inc., “DO-178B: Software Considerations in Airborne Systems and Equipment Certification,” December 1992.

[33]RTCA Inc., “DO-178C: Software Considerations in Airborne Systems and Equipment Certification,” December 2011.

[34]RTCA Inc., “DO-333: Formal Methods Supplement to DO-178C and DO-278A,” December 2011.