Abstract—Intelligent agents applied to the control of Unmanned Aerial Vehicles (UAV) is becoming a much discussed line of research recently, as an approach to increase autonomy. In this paper, we present UAVAS (Unmanned Aerial Vehicles AgentSpeak), a framework that employs AgentSpeak agents to reason about the courses of action to take based on perceived information from the environment, thus allowing the programming of missions for teams of UAV in this high-level language. The UAVAS framework derives from the AgentSpeak runtime implemented in Jason and aims to allow the control of UAV behaviour to be programmed using the multi-agent oriented paradigm.

I. INTRODUCTION

Although UAVs have been around for a long time, it is only in the last decade that great advances have brought them to the fore. Some of these advances are in the field of artificial intelligence and complex behaviour development where multi-agent systems are taking a leading role in some of the research around the world, for example in path planning, coherent/coordinated teamwork, free-flights management (autonomous UAVs), and others.

To support all of these scientific and technological developments, there is significant need for IDEs, platforms, and tools built specifically for the development of software for controlling the top-level behaviour control of UAVs. To the extent we looked into the literature, there seems to be a clear lack of highly abstract languages suitable to the development of complex (possibly AI-based) behaviour for UAVs. Typically, the main languages used to code software for UAVs are low-level ones such as C or Assembly. The fact that UAVs require complex technologies to be built just makes it more challenging for potential practitioners to develop applications for UAVs.

We are not aware of any open-source framework or language that supports code development for UAVs with highly abstract programming languages. There are cases where an agent oriented language has been used in projects involving UAVs, like JACK\(^1\), which is an advanced agent-oriented framework, but the problem here is that it is a commercial tool.

To address the lack of such a framework, we are taking steps towards developing a platform (composed by a runtime engine, a bridging protocol, interfaces and extra hardware), to serve as a technology bridge between existing UAVs and the AgentSpeak agent-oriented language. This framework is to be used in AgentSpeak programming to refer to existing features of the firmware available within the UAVs. This is done through the mapping of actions to methods and signals to percepts, so that the virtual environment for an agent serves as an interface to/from the UAV electronics. AgentSpeak is an agent-oriented language\(^2\) with an open-source implementation called Jason\(^3\) developed in Academia, with some documentation and to some extent formal semantics too.

The software development and experiments for this project are on-going, and we are currently considering also hardware/software compatibility issues, in regards to specific hardware used in the Mikrokopter and the extra board we hope to add to a couple of UAVs to which we have access. The Mikrokopter project is an open-source project, with firmware available to public access at [http://gallery.mikrokopter.de/](http://gallery.mikrokopter.de/). Further things we are currently considering in regards to our UAV project is the use of speech-act based communication and agent-oriented programming theory to develop teams of UAVs in the context of applications that require a number of UAVs to work in cooperation or in a coordinated way.

The remainder of this paper is divided into 5 more sections. Section 2 provides a brief overview of recent research on UAVs and some current issues. Section 3 discusses the research problems we aim to address and our proposed approach. Section 4 gives a brief introduction to the Mikrokopter project for UAV technology. Section 5 presents the current architecture of the framework we are developing and some of its features. Section 6 shows a preliminary case study in applying the UAVAS framework: the programming of a UAV to take a photograph at a given location in a planetary exploration scenario. Finally, in Section 7, we discuss the current state of our work, some conclusions and future work.

\(^1\)JACK Intelligent Agents, or simply JACK, is a framework in Java for multi-agent system development.

\(^2\)AgentSpeak

\(^3\)Jason
II. RECENT RESEARCH

The multi-agent/team behaviour reference is very common in UAV research. The analysis of coordination between coalitions \([1]\) and large-scale dynamics \([5]\) are research efforts to establish a more predictable system in chaotic systems with large numbers of agents. Path planning is another issue often mentioned, using various techniques from graph theory \([6]\) and heuristics, both for unpredictable environments as well as for well-defined ones \([7]\). One specific topic of research in path planning is the issue of visual-field optimisation and efficiency \([8]\) with its main concepts based on trigonometry.

The research in algorithms for avionics took important space in the academic arena. Recently, there has been significant research efforts in the free-flight category of UAVs, where the avionics is capable of making decisions about their own paths, in a dynamic way. In \([9]\), the simulation of free-flights control/manager is a mentioned issue, with analysis of flight control and path planning. The flight plans are made by each UAV’s avionics individually, with a contextual intelligence acquired through AI techniques. The system’s behaviour is studied and collision avoidance algorithms are implemented inside each UAV node, in a specific framework described in \([10]\). Also, for this same issue, there is an approach in which deconfliction for collision avoidance is treated as negotiation between the UAVs \([11]\).

Other researchers, e.g. \([12]\), developed a more formalistic view of multi-coordinated path-planning, with mathematical models used for finding optimal solutions. To enable testing of existing UAV software, a testbed research laboratory described in \([13]\) has been equipped with motion capture sensors, allowing the treatment of UAV moves to be controlled by high advanced image-processing software. Using this same system of sensors, an approach has been developed where the scenario is a 3D city \([14]\), in which a major objective is the coverage of the 3D structures.

Also, other research topics that can be mentioned are autonomy, risk analysis, see-and-avoid systems, or networked coordination, being investigated in a significant number of laboratories all around the world, of which we can mention Grasp Lab \([2]\) (Pennsylvania), Santos Lat \([3]\) (Brazil), Centaurus Technology \([4]\) (Malaysia) and the SiDeV AAN group from Universidade Federal de Minas Gerais.

As we can see, AI techniques for UAV control and team behaviour for UAVs are both becoming popular. The main idea in all the research mentioned above is the use of specific AI techniques for UAVs. In this type of work, there are so many barriers and difficulties. Furthermore, it was only in the last decade that common people could start acquiring some of the UAV prototypes as prices became more accessible (although still most often not so) and additionally, the technologies, software, and general information about UAVs became more accessible. Autonomy for UAVs is not an easy research area, as there are so many obstacles that researchers typically find. Some of the reasons for the current difficulty in doing research and development for UAVs are as follows.

- Recent Technology: As seen earlier, UAV is a recent technology, and most often developed for (confidential) military applications, which makes it rather hard to find documentation, open-source projects or well developed frameworks for the development of specific applications of UAV technology.
- Costly Hardware: In most cases, boards, software, and electronics are hard-to-find and very specific. Generally, a single UAV prototype shows an overwhelming cost (even if their costs have dropped significantly in the last decade) and very few researchers can afford to buy UAVs.
- Multi-disciplinary Research Field: UAVs are constructed based on multi-technological knowledge, which involves aeronautical engineering, electronics, computer science (e.g., image processing, embedded systems development) and others. This great number of skills means it is hard to do significant progress within small groups of researchers or without direct governmental support.
- Specific Areas for Tests: We cannot test UAV applications in open civil areas due to the dangers that the prototype can offer in some cases. Closed labs or open uninhabited areas are generally the best choices to test these devices. Also, in some countries, the researchers need special permissions to use UAVs.
- Software complexity: in order to program specific behaviours for those devices, it is necessary to have a background in particular low-level programming languages, such Micro-C or Assembly. The code that defines the behaviour of UAVs are not at all simple to write or maintain when written in such languages.

Yet, others issues such as airworthiness, certifications, complex control architectures, energy efficiency, fail-safe systems, payloads care, navigation systems, security, smart sensors, safety/security, system integration, and regulations are challenges which needs to be solved. An enormous background must be obtained to enable someone to fully develop UAV technology.

A. Multi-Agent Oriented Programming

A topic that only very recently has started to appear in the UAVs area is the use of the multi-agent oriented programming paradigm applied to UAVs. Multi-Agent System (MAS) is a sub-area of Distributed Artificial Intelligence \([15]\) that focuses on the study and research of software that is developed through a number of autonomous, interacting entities who share the same environment. In a MAS, each agent is one of its active entities, where an (organised) number of such entities form a multi-agent society. In the UAV context, an agent can be viewed as a UAV and many interesting experiments can be carried out with this approach. Usually, each agent has a set of behavioural capabilities, a set of goals and some autonomy (possibly also some intelligence) needed to make choices about those behavioural capacities. Decisions on what

\[\text{http://www.grasp.upenn.edu/}\]
\[\text{http://www.santoslab.com.br/index.htm}\]
\[\text{http://www.centaurustechnology.com/}\]
action to take are determined taking into consideration the changes perceived in the environment and the desire to achieve long-term goals. For further details about multi-agent oriented programming see [16], [17].

III. LACK OF TOOLS AND THE ABSTRACTION GAP

In the literature survey that we have conducted, we have not been able to find any open-source frameworks, based on agent-oriented languages, that were tailored specifically to the context of UAVs, not even in academic projects. Clearly, coding all of the behaviours expected of UAVs in complex applications in low-level languages is an extremely hard task for programmers. From this point of view, our focus with this research is to create an open-source tool to help the development of UAV applications, with a customised environment and IDE. AgentSpeak comes in hand because it is based on agent programming (thus facilitating the development of autonomous software that needs to react to a changing environment while attempting to achieve users’ long-term goals), has formal semantics and development tool support. Also, it is worth mentioning that it has a free (open-source) implementation called Jason [3], [18]. Jason is an open-source interpreter for an extended version of the AgentSpeak language, developed on top of Java technology, and has been improved, extended, applied and studied by the academic community in the last decades. Through an integrated API and an IDE with a deliberative multi-agent runtime, multi-agent systems can be implemented and deployed using a very high-level language for specifying agents’ overall practical (i.e., action oriented) reasoning, opening up the possibility to research several topics of current interest in computer science, including avionics and UAVs. Some of the existing work that uses Jason/AgentSpeak is the research on virtual environments reported in [19] and teams for an agent programming competition.

AgentSpeak is an agent-oriented logic-based language used to program multi-agent systems and complex agent behaviour. A brief description of this language is given below.

A. AgentSpeak: Reasoning about Courses of Action

AgentSpeak is a very popular agent-oriented programming language initially conceived to bridge the gap between theoretic and practical work in BDI Agents research [2]. Its best known implementation is Jason [3]. An AgentSpeak agent is defined by a set of beliefs that forms the initial state of the agent’s belief base, which is a set of logical atomic formulae (first-order), and a set of plans that will be part of the plan library in the agent’s initial state. An AgentSpeak plan has a head that consists of a triggering event (specifying the events for which this plan is relevant), and a conjunction of belief literals representing a context. The conjunction of literals in the context must be a logical consequence of current beliefs that the agent has for the plan to be considered applicable at the time a plan is being chosen to handle a particular event (only applicable plans can be chosen for execution). The plan also has a body, which is a sequence of basic actions or sub-goals that the agent must perform to successfully handle the event that triggered the plan; events are either changes in beliefs or changes in goals of the agent. Basic actions represent atomic operations that the agent can do to alter the environment. These actions are also written as atomic formulae, but using a set of action symbols instead of predicate symbols. AgentSpeak distinguishes two types of goals: achievement goals and test goals. Achievement goals are formed by an atomic formulae prefixed with the operator {+}, while test goals are prefixed with the operator {-}. Plans are triggered by the operators {+} (addition) or {-} (deletion) of beliefs or goals.

B. AgentSpeak Applied to UAV

Thinking about the problems in UAV research pointed out earlier, our focus is in the direction of applying Multi-Agent Systems techniques to UAVs, in particular to support software development for applications where UAVs need to work as a team (in a coordinated, cooperative fashion). Our main objective is to create a framework for UAV behaviour programming based on a multi-agent oriented programming language, which could provide more expressivity, through programming with high-level abstractions. In this same topic, there are some other on-going research efforts. Examples that can be mentioned are the Automated WingMan [20], where a framework based on JACK agents has been developed for using agents to make the high-level decisions on the actions UAVs should take. While they apply the BDI model with the same purpose as we do, their work is not openly available.

A more general approach for software development for controlling UAV behaviour, based on collaborations controlled by hierarchical rules, is reported in [21]. They use techniques such as task communication/exchange, path planning, and breaking problems into sub-tasks.

One could ask the question: why use AgentSpeak? AgentSpeak is a well developed agent oriented language, with formal operational semantics and a free-open source IDE (Jason). There are other interesting agent programming languages such as JACK, 3APL, or Brahms but, in our point of view, none of these languages match some of AgentSpeak’s characteristics mentioned above, such as fully operational open-source implementations and good documentation, while based on solid theoretical work.

Below we show some C code from the Mikrokopter project (described in next section) to give an example of the low-level at which most UAV applications are normally programmed. The following snippet adds a new waypoint to be reached by a UAV prototype.

\[\text{http://www.multiagentcontest.org/}\]
C Source

```c
u8 WPList_Append(Waypoint_t* pwp) {
    if (WPNumber < WPLISTLEN) {
        memcpy(&WPList[WPNumber], pwp, sizeof(Waypoint_t));
        WPNumber++;
        NaviData.WaypointNumber = WPNumber;
        return TRUE;
    } else return FALSE;
}
```

Code for Adding a New Waypoint.

In view of the significant gap between ideal behaviour programming and C/Assembly coding, our objective is to minimise such abstraction gap and to create an IDE for developing complex behaviour for teams of UAVs. More specifically, we propose a framework to allow programmers to write code in AgentSpeak for the development of UAV systems, assisted by modern multi-agent oriented programming.

IV. THE MIKROKOPTER PROJECT

The MikroKopter is a technology project that supports the deploying of propelled UAV prototypes with various numbers of propellers and various possible configurations. In all these configurations, there are three board modules in common, each of them doing a specific task. These boards are:

- **Flight-Ctrl**: The **Flight-Ctrl** board takes care of the system and environment measurements such as angular velocity of the axes, acceleration, atmospheric pressure, battery charge, processing and computing angular position and electronic speed controllers. This board is distributed in versions v1.0 (green board) and v1.3 (red board). It contains an 8 bit microprocessor Atmel ATMEGA644 20MHz processor.
- **Navi-Ctrl**: This board extends the functionalities of the **Flight-Ctrl** with a GPS system and coordinates management. The **goHome** and **holdPosition** actions (explained below) are available from this hardware.
- **Brushless-Ctrl**: This board can be seen as an update since it diminishes interface errors, interference risks, and improves efficiency and performance, managing the brushless control engine and being activated by PWM (Pulse Width Modulation);

Each board works with a signal protocol which is used by the system to “decode” and manage the programmed instructions. In Figure 1 a four-propelled mikrocopter is shown.

Some recent work related to this prototype can be found in [22], [23], [24].

V. THE PROPOSED FRAMEWORK

Working on Mikrokopter technology, our aim is to deploy an agent platform within a Mikrokopter prototype, which will serve as a runtime for the agent system. This platform uses the AgentSpeak/Jason architecture, interfacing it to the UAV control boards through a specific communication protocol. This protocol is intended to offer signal decoding features specific to the mikrokopter internal interface layer, providing percepts (resulting from sensing the external environment where the UAV is situated) to the reasoner. This is done by implementing external actions available to the AgentSpeak agent. We also adapt speech-act based messages to serve as direct UAV-to-UAV communication. The purpose of the protocol is to work as a technology bridge between AgentSpeak/Jason and the UAV. To run the system, additional hardware is intended to be plugged into the prototype with the task of managing the high-level decision carried out by AgentSpeak agents, exchanging signals with other hardware components via USB and/or serial port. In summary, the AgentSpeak agent’s main purpose is to provide some level of autonomy to the prototype. A closely related work that we can mention is [25], where MathLab tests of an auto-pilot mechanism have been carried out. Figure 2 depicts the overall idea of how UAVAS is to work, and Figure 3 shows the system engine (for a single UAV). The body-level represents the UAV components as originally purchased, composed by the Flight-Ctrl, Navi-Ctrl, and Brushless-Ctrl boards. To support the agent platform, an operational system that runs Java will be used, most likely.

Fig. 1. Mikrocopter Configured in a Four Propeller Set.

Fig. 2. Overall View of UAVAS

Fig. 3. System Engine
Mobile agent on which Jason runs directly. Taking a closer look at the agent runtime engine, it encapsulates functionalities from encode/decode signals to actions/percepts. This way, the agent platform can manage the perceptual information and reason about the best protocol action to take. The current version of the protocol is informally described below.

The Bridging Protocol

- `moveTo(Point)` – This action adds a new location where to move, on the shortest possible path.
- `doPatrol(ListOfPoints)` – When this action is performed, for each point in the given list, and beginning at index 0, the agent executes the `moveTo` action, assuming as argument the indexed points in the list. When the position in last index is reached, the index is restarted to 0 and the UAV carries on patrolling all those positions.
- `goHome` – Go to the pre-defined home point location, using the firmware method for that.
- `setHome(Point)` – Set the given point as the home position.
- `holdPosition` – Hold position at the current location.
- `addWayPoint(Point)` – Add a waypoint location, using the firmware method for that.
- `request(ToAgent,Literal)` – Send a request to another agent, where the passed literal is assumed to be false, and the other agent is requested to try to make it true.
- `inform(ToAgent,Literal)` – Send to another agent the given literal, with the intention that the receiver will believe the literal is true.
- `ack(ToAgent)` – This action is used to acknowledge that a request has been received.
- `checkBattery` – Send a query to Flight-Ctrl to retrieve information (to be acquired as percepts) about battery charge.
- `checkLocation` – Send a query to Navi-Ctrl to retrieve information (to be acquired as percepts) about the GPS location.

These actions are not just procedures called by the UAVAS engine but Jason external actions that are called by UAVAS agents in AgentSpeak programs. There are three main groups of action: logistic, communication, and sensing. The logistic actions are used to change the UAV positioning controlled by GPS in some way. Communication actions make it possible for agents (in our context, the UAVs) to send messages to each other. Sensing actions are used to acquire information about the environment or the UAV conditions. Also, the prototype’s remote controller (which is used to control the avionics) is assumed to have priority over UAVAS operations due to system safety issues.

VI. EXAMPLE: PROGRAMMING AN UAV AS AGENT TO TAKE A PHOTOGRAPH

In this example, we show part of a code where a photograph mission is implemented using the UAVAS framework. The UAV’s main objective is to perform the following sequence of steps:

- Check the systems, to obtain the necessary data about environment, battery, and location;
- Ensure that the travel conditions are met;
- Initialise a cruise to the target position for this mission;
- When at the right location, take a photograph;
- Turn back home.

Our code example follows:

```
// Rules
// ========
// calculate distance
distance(LtP, LnP, LtC, LnC, HtP, HtC, D) :=
  D = sqrt((LtP - LtC)^2 + (LnP - LnC)^2 + (HtP - HtC)^2).

// check if can safely reach a location from current position
can_reach(LtP, LnP, HtP) :=
  fuel(F) & at(LtC, LnC, HtC) &
  fuel_efficiency(E) &
  distance(LtP, LnP, HtP) <= D/F.

// Initial Goals
// ==============
!initialize. // to update beliefs
!mission. // today’s mission

// Plans
// =======
! initialise : true <-
  checkEnvironment;
  checkBattery;
  checkLocation;
  setHome(-30, -51.1, 0).

!mission :
  system(ok) <- // believes the systems are all OK
  photograph(-23.36, -48.07, 100);
```
reach((LtP, LnP, HtP))

reach((LtP, LnP, HtP)) ←
moveto((LtP, LnP, HtP));
!at((LtP, LnP, HtP));

shoot. // extra action, to take the photograph

// the conditions for the mission have not been met
¬photograph((LtP, LnP, HtP)) ←
goHome.

// used to check if the mission position has been reached
+!at((LtC, LnC, HtC)):

!at((LtP, LnP, HtP)) &
(LtP <= LtC | LnP <= LnC | HtP <= HtC) &
can_reach((LtP, LnP, HtP)) ←

wait(100); // wait 1 second

+!at((LtP, LnP, HtP)); // check again if already there

+!at((LtP, LnP, HtP)). // the agent believes it is there

The first part of the code (Prolog-like rules) has two rules: one to calculate the distance between two geographical coordinates used by the UAV reasoning agent to find the distance between its present location and a given destination, and another to check whether it appears viable for the UAV to go from its present position to a target one. The initial behaviour of the agent is determined by its two initial goals. The first is used to initialise the agent state through the +!initialise plan which calls the UAVAS methods to retrieve percepts about the UAV current state and information about its location, updating the agent’s belief base by means of the agent belief update internal mechanism. The +!photograph plan is used to check, based on the updated information about the environment, if everything is OK, including whether the target is at a distance the UAV can reach. If everything is OK, it adds a new waypoint to the UAV (via the moveto operation) which is sent from the virtual environment as a signal to the UAV body. It then needs to make sure that, through the waypoint adding action, it has successfully achieved the goal of being at the target coordinates. If the location has been successfully reached, it takes the picture and finishes off the mission by the goHome action. This external action, internally to UAVAS, just adds a waypoint to go to the coordinates initially set as “home”.

This is, of course, a very simple example, where detailed cruise-time issues have been ignored, but those could also be taken into consideration in a realistic application without major changes in the part of the code presented above. The code shown above is only meant to illustrate the approach, and to show that it has relatively high abstraction and readability in comparison to low-level languages (such as C or Assembly). One can easily picture that an implementation of this same autonomous behaviour would be much more cumbersome if coded in traditional programming languages.

VII. CURRENT AND FUTURE WORK

Currently, we are working in three main directions:

• UAV communication protocols for AgentSpeak. The protocols are based on a set of speech-act “performatives” in the KQML style8 to enable AgentSpeak agents to controlling a team of UAVs to communicate and coordinate at mission time.

• UAV-team Simulator. To test the agent behaviours developed in AgentSpeak, we are working on implementing a simulator where the environment is a virtual airspace and the possible operational commands passed on to the simulated UAVs comes from the protocol defined in this paper.

• Android-based agent platform. The main goal here is to develop a well-suited base where the agents can run and receive/send messages, with low energy consumption. Some of the concerns are the implementation of a USB/Serial data link to send/receive instructions/information from the agent platform to the UAV prototype.

Currently, we are trying to make this initial framework run on the Beagle Board9 hardware attached to the UAV prototype, on which the UAVAS framework with the Jason runtime for AgentSpeak agents runs. The main platform chosen to implement the first version of the proposed framework is the Google mobile platform Android, because of its compatibility with the Beagle Board, Java technology, and low energy consumption. In Figure 4 we show the various software layers in our current proposed approach.

![Fig. 4. UAVAS Software Layers](image)

Also, for debugging purposes, at this initial stage we are implementing console-based experiments in an Android emulator. Our intention is to deploy UAVAS using only Android’s basic hardware management such as USB/Serial ports to do the data exchange. In Figure 5 we show the debugging console that we are currently using to test the protocol actions being implemented (still under development). Below, we show a snippet of a Jason customisation code for the UAVAS moveto action which receives a WayPoint as argument (referring to a GPS position) and, after the arguments are checked, passes it on to the UAVAS interface which then translates it to the

8The Knowledge Query and Manipulation Language, or KQML, is a language and protocol for communication among software agents and knowledge-based systems [26].

9Beagle Board is a low-power, low-cost single-board computer produced by Texas Instruments in association with Digi-Key.
appropriate signal to be sent to the prototype hardware (via USB or Serial port).

![UAAS Engine Console](image)

```
// Update the "patrol" belief
removePerceptsByUnif(
    agName, Literal.parseLiteral("patrol(_)")
);  
addPercept(agName, Literal.parseLiteral("patrol(no")));  
// Call firmware method [Send target Position],  
// with the passed point as argument
double lat = Double.parseDouble(action.getTerm(0).toString());  
double lon = Double.parseDouble(action.getTerm(1).toString());  
double hei = Double.parseDouble(action.getTerm(2).toString());  
bridgeConnection.move_to(new GeoPoint(lat, lon, hei));
```

**Fig. 5.** UAAS Debugging Console

The selection of the Android operating system was made with our objective to use open-source tools in mind. Although AgentSpeak/Java is a free IDE, currently there is no plugin or implementation that makes it possible to use it on others Java platforms such as J2ME, a more suitable basis for mobile software. For now, the Android platform is useful because of its technological compatibility, light-weight runtime, and open source API. Also, for this first version of UAAS, we are not taking into consideration things like energy/battery consumption rates. These and various other issues have been left for future work, where this research could also take several others directions. Some possible directions are in the implementation of plugins for Jason supporting mobile technologies, or the development of an IDE for a UAV-specific AgentSpeak based language. Such efforts could potentially lead to a framework that is technologically more suitable for mobile and light-weight applications.

### ACKNOWLEDGEMENTS

The authors acknowledge partial financial support for this research from CNPq and FAPERGS.

### REFERENCES

