An Agent-oriented Ground Vehicle's Automation using Jason Framework

Reydson Schuenck Barros, Victor Hugo Heringer, Carlos Eduardo Pantoja,
Nilson Mori Lazarin and Leonardo Machado de Moraes
CEFET/RJ, Gov. Roberto da Silveira 1900, Nova Friburgo, 28635-000, RJ, Brazil

Keywords: Unmanned Vehicles, Multi-agent Systems, Robotics.

Abstract: This paper proposes an agent-oriented ground vehicle automation that uses low-cost hardware. The vehicle's platform consists in a group of hardware and software layers that acts with the Jason programming language for unmanned vehicles automation. This paper also presents a methodology with four programming layers to facilitate the hardware integration and implementation. To validate and demonstrate the platform an unmanned ground vehicle was constructed using an ATMEGA328 microcontroller, a library for serial communication and a six-function remote controlled vehicle. The vehicle is able to move from one point to another based on its global position.

1 INTRODUCTION

An intelligent agent can also be as physical as virtual and has the capability of acting upon a simulated or a real environment. Besides, it has individual goals and can communicate with others agents in order to satisfy its goals (Ferber, 1998).

Recently, a new applicability of Multi-Agent Systems (MAS) is the development of autonomous unmanned vehicles systems. These vehicles do not need embedded pilots and, in many cases, are guided by a portable or mobile control station. In this process, a group of problems can be identified like human failure and communication that interferes the vehicle's mission.

There are several frameworks that use the agent-oriented approach and looks for the Unmanned Aerial Vehicles (UAV) automation, like (Wallis et al., 2002), that uses JACK intelligent agents (Busetta, 1999) to provide a simple programming environment of flying tactical behaviors; and (Huff et al., 2003), that developed a simulator capable of representing an UAV agent with different flying planning approaches. However these works do not present hardware connections, acting only with simulations.

There are some platforms that embed MAS into a specific hardware, e.g. (Karim; Heinze, 2005), which provides an agent architecture programmed in JACK for the *Codarra Avatar*. However, the platform do not allow other UAV integration and the vehicle needs to be manned until reach a safe altitude. In (Hama et al.,2011), the UAVAS platform uses the Jason (Bordini et al., 2007) framework that is a set of hardware and software used for the *Microcopter* quadricopter automation. Despite the idea of a generic architecture and the possibility of others firmware integration, this platform is specific for aerial vehicles.

So, the objective of this paper is to propose an agent-oriented ground vehicle that uses a set of low-cost hardware and software layers supported by the Jason framework. Besides, the vehicle's platform can be used with any vehicle (aquatic, aerial and ground). To exemplify the vehicle's operation will be used: an ATMEGA328 microcontroller; the RXTX library for serial communication, and a six-function remote controlled vehicle.

This paper is structured as follows: in Section 2 will be presented the Jason framework basic concepts; in Section 3 will present the physical structure of the platform and will also present a programming methodology used in the vehicle's automation; in Section 4, a functional example will be developed with the proposed vehicle's platform; in Section 5 some related work will be analyzed; and finally, in Section 6 a conclusion and the related works will be presented.

Schuenck Barros R., Hugo Heringer V., Eduardo Pantoja C., Mori Lazarin N. and Machado de Moraes L..
An Agent-oriented Ground Vehicle's Automation using Jason Framework.
DOI: 10.5220/0004917102610266
In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence* (ICAART-2014), pages 261-266
ISBN: 978-989-758-016-1

2 JASON BASIC CONCEPTS

In this section will be presented some Jason framework concepts that will be used in the platform layers. The Jason agent-oriented framework is based on the AgentSpeak and Java languages for MAS development that uses the behavioral model Belief-Desire-Intention (BDI) (Bratman, 1987). In Jason an agent can be implemented using beliefs, goals, plans and actions. An agent can acquire beliefs perceiving the environment and communicating with others agents. The goals are mental states that agents desire to achieve in order to modify their environment. These goals are activated based on the beliefs perceived about the environment (Wooldridge, 2000) and can be represented by plans in Jason.

A plan is composed of three different parts: (i) the trigger event; (ii) the context; and (iii) the body of a plan. The trigger event is responsible for the plan activation, while the context is used to define the applicability of a plan, satisfying to a certain condition. The body of a plan is a set of actions that an agent has to execute to achieve a goal.

In Jason an action is executed when a plan is activated and can be of two types: (i) the actions that change the environment and are executed outside the agent through Java methods (these methods are implemented to represent the effective action in the environment); (ii) the internal actions that can be pre-defined functions or communication function created to support the agent reasoning (Bordini et al., 2007).

3 THE VEHICLE'S PLATFORM

This section presents the vehicle's platform that was projected as a sequence of hardware and software layers that can be embedded in order to provide vehicle's computational intelligence and autonomy. The platform is composed of five layers: the *Hardware*; the *Firmware*; the *Operational System (OS)*; the *Simulated Environment*; and the *Reasoning Agents*. The platform architecture can be seen in figure 1.

The first layer, the *hardware*, contains (i) the vehicle that needs to be automated; (ii) the actuators that convert the electrical signals into actions in the real environment; (iii) the sensors that intercept the environment's information and convert them into electrical signals for the microcontroller layer. In this layer it is chosen the vehicle type which will be automated, as the sensors and the actuators that will be used.

In the second layer, the *Firmware*, the sensors are controlled by a microcontroller which converts the electrical signals into data that can be transmitted by a serial communication. This layer generates electrical signals to the actuators, localized at the vehicle, based on the data received by the serial communication. In this layer, the sensors are coupled at the microcontroller board, which in turn it is coupled at the vehicle's hardware with the actuators.

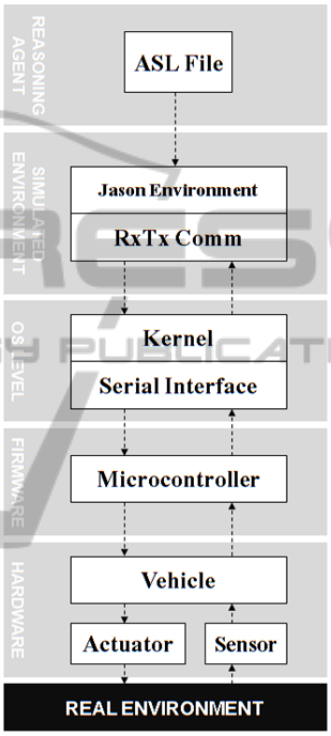

Figure 1: The vehicle's platform.

The third layer, the *Operational System*, is the serial interface, which through the *Kernel,* buffers the data that arrives at the serial port or send the available data to the agent's environment when solicited. In this layer is chosen the OS that is responsible to keep the next layers operation.

The fourth layer, the *Simulated Environment*, contains the agent's environment which receives its actions commands. The environment is composed of the modified *RxTx Library* working along with the Jason Environment Class. The *RxTx Library* is responsible for the connection between Java language and the *Kernel* serial interface and allows the both side data transmission through the serial port. It still receives the external actions that need to be executed by the device into the real environment.

The last layer, the *Reasoning Agent*, is

responsible for all the platform cognitive reasoning and is programmed in Jason, which through plans, goals, beliefs and actions is capable of controlling, in real-time, the vehicle. The Jason framework uses the BDI paradigm that adds behaviors aspects of human-being reasoning.

The platform permits any hardware automation that can use a microcontroller and this work focuses on autonomous vehicles. The platform allows the low-cost hardware integration and uses a set of free software that is platform-independent, making the solution accessible for many objectives.

3.1 The Platform Methodology

To use and implement the vehicle's platform is necessary to follow a specific development methodology in order to integrate correctly the several hardware-software layers. There were identified four programming layers that need developers' intervention: the *Hardware*; the *firmware*; the *Simulated Environment*; and the *Agent Reasoning*. The layers hierarchy can be seen in figure 2.

Initially, a list of all agents' environment perceptions is necessary, as all actions that need to be executed by the agents. From the list, the necessary sensors and actuators are chosen, and they have to be compatible with the microcontroller. In the *Hardware* layer, the necessary voltage and amperage for the motors activation must be analyzed (if the microcontroller does not supply those needs directly). In case of the power supply needs to be in different polarities, it will be necessary adequate H bridges and power transistors.

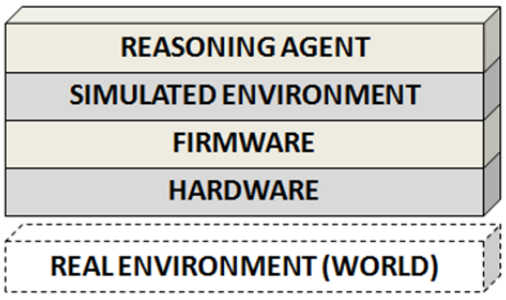

Figure 2: The four-step methodology.

In the *Firmware* layer, the microcontroller must be programmed to receive serial communication data and to send commands to the actuators from these received data. The sensors data need to be interpreted and to be sent to serial communication too. Besides the sensors and actuators data, the physical communication between agents must be programmed in this layer. After the firmware codification, it is necessary the code compilation and upload it to the microcontroller.

In the *Simulated Environment* layer, the environment is programmed in Java language where it is necessary to: (i) program the interface serial integration with the agent's environment using the *RxTx Library*; (ii) update the agent's belief base from the microcontroller data; and (iii) send the agent's external action to be executed by the microcontroller. The methods responsible for the logical communication between different vehicles must be programmed in this layer.

Finally, in the *Reasoning Agent* layer, the agents are programmed using Jason framework, where the plans collection (with their respective actions) that an agent can execute based on its beliefs, desires and intentions are inserted.

The methodology presented allows the use of several independent abstraction levels by the developer. It starts with the hardware choice and analyze until the vehicle's cognitive programming. The layers are integrated from a series of communication and data flow methods. Thus, it is possible to reutilize existents architectures, microcontrollers and libraries for unmanned vehicles automation or begin a new prototype, if it is desirable.

4 THE VEHICLE PROTOTYPE EXAMPLE

This section presents a simple working example using the vehicle's platform. The example consists of a ground vehicle that moves from one point to another based on its global position. The initial point is obtained by the GPS device installed in the vehicle, while the end point is informed manually. For demonstration it will not be considered any obstacles between the initial and the end point. The example uses the ATMEGA328 microcontroller; the *RxTxComm* library for the serial communication that is free; a Pentium IV computer; and the Jason framework. The platform is embedded into a six-function remote controlled vehicle. The vehicle can be seen in figure 3.

The first level of the methodology is to analyze the chosen hardware. The electrical signals that activate the vehicle's motors were analyzed and the GPS device was chosen too. Besides, a group of hardware components were used to facilitate the vehicle's automation.

Afterwards, the microcontroller must be programmed with functions that activate the motors forward, backward, left and right. It is necessary to program the functions that will receive the serial communication data from the virtual agent environment.

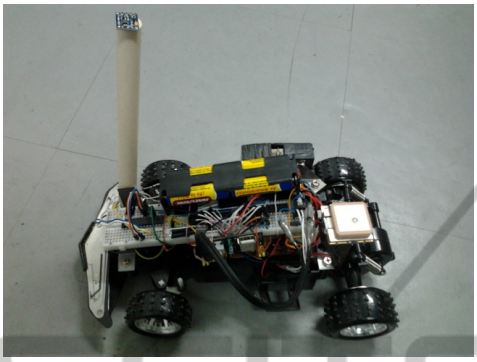

Figure 3: The vehicle prototype.

The function that gets the position from the GPS and sends to the virtual agent environment still needs to be programmed. The function *sendGPS* gets the data from the GPS device and send it to the serial port was implemented.

The loop function, responsible for the microcontroller cycle, calls the *sendGPS* whenever the GPS data is requested; the *guide* function to control the vehicle; and reads the buffer information.

The next step of the methodology is to program the virtual agent environment in Java integrated with the *RxTxComm* library that is the bond between Java language and the microcontroller. The Jason environment is responsible for the methods that represent the agent's actions and the serial communication methods.

In the simulated environment was developed a class library that is responsible to guide the vehicle using the earth's pole as point of reference. So, the reasoning agent will be able to analyze its own movements based on the processing of the GPS data. The class library allows a route analysis based on two global position points and provides data indicating if the vehicles deviates from its original path.

To develop the library class was considered the route as the least distance between two points into a sphere. Considering the Earth's rounded format and a planet's pole as reference, every route deviation forms a spherical triangle that can be analyzed. A spherical triangle can be seen in figure 4.

After some geometric analysis with spherical triangle's properties and the law of sines and cosines

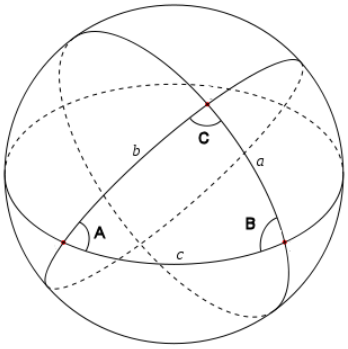

Figure 4: The spherical triangle.

a equation for calculate the distance between two points into a sphere with the Earth's radius was used (Milone; Wilson, 2008). The equation can be seen below:

$$\widehat{AB} = cos^{-1}[sin\varphi_A sin\varphi_B + cos\varphi_A cos\varphi_B cos(\gamma_B - \gamma_A)]r \quad (1)$$

With the distance's equation defined it was possible to find an equation to correct the trajectory of a body. In this case, if the vehicles goes out of its trajectory, the equation returns the deviation's angle from its original points. Afterwards, the agent will be able to perform a correction movement based on the deviation's angle. The angle's equation can be seen below:

$$\alpha = cos^{-1}\left[\frac{sen\varphi_B - sen\varphi_A cos\widehat{AB}}{cos\varphi_A sen\widehat{AB}}\right] \quad (2)$$

Others equations were developed to support the movement definition: i) an equation to correct the quadrant angle value because the angle's equation returns always the lower possible value; and ii) a stop equation in meters that defines a stopping area based on the GPS latitude and longitude of the destination point.

The last step of the methodology is to program the virtual agent using the Jason framework. The agent reasoning is programmed based on beliefs, plans, intentions and actions. The agent changes its beliefs from the perceptions of the simulated environment. The agent request the real world perceptions by performing an action called *getPosition*. The GPS data will be transferred from serial port to the simulated environment and after the movement library process the received data, the results will be transferred to the agent mind as perceptions.

The actions *goAhead*, *goBack*, *goLeft* and *goRight* will send a char from serial port to the microcontroller and will activate the motors to perform the selected movement. The *stop* action will

stop the vehicle sending a char that will deactivate all motors. For last, the *align* action will align the vehicle deactivating the left and right motors. The agent codification is shown below:

```
!start.

+!start : true <-
   +running;
   !getPosition;
   !move.

+!getPosition : running <-
   getPosition.

+!move : running <-
   goAhead;
   !getPosition;
   !move.

+!move : running & arrived <-
   -running;
   stop.
```

The agent reasoning is basically composed of plans and actions. The plan *start* begins the reasoning adding a *running* belief at agent's belief base and calls the *getPosition* plan to obtain the current position from the GPS device. Afterwards the *move* plan is called and will be executed only if exists the *running* belief.

The *getPosition* plan performs the *getPosition* external action while the *running* belief is in the agent's belief base to update the vehicle's current position until it reaches the destination point. The *move* plan will have two different behaviors depending on the trigger event: (i) if the vehicle is running and position was not reached yet; and (ii) if the vehicle is already running and the pre-defined position was reached (represented by the *arrived* belief). In the last case, the *running* belief is removed from the agent's belief base, stopping the movement. The *arrived* belief will be added by the movement library when the vehicle reaches the pre-defined stopping area.

It was realized an experiment with the vehicle prototype between two points within a distance of ten meters. The vehicle could move forward to the destination point correctly and stopped inside the pre-defined stopping area of 1 meter. Besides, this simple example proves that the platform can automate any vehicle because the movement library was developed to support all kind of vehicles. The automation can be performed using just low-cost hardware and free software as can be seen above. The vehicle could perform all programmed movements using a Jason agent that was responsible for all the reasoning based on its real world beliefs.

5 RELATED WORK

There are some works that propose platforms for unmanned vehicles using agent-oriented programming languages, frameworks and methodologies. In (Wallis et al., 2002) a simulation platform is proposed for unmanned aerial vehicles. The platform uses the JACK intelligent agents but it did not present hardware integration, working only with simulations. In (Huff et al., 2003), a simulation environment for different flight approaches is proposed, but it not integrates any hardware in simulations.

In (Karim; Heinze, 2005), an architecture for JACK agents uses the UAV *Codarra Avatar*. But the platform does not permit other hardware selection, binding the platform with the selected vehicle. The vehicle still needs to be manually piloted until reaches a safe altitude and then the agent controls entirely the vehicle.

The UAVAS platform proposed by (Hama et al., 2011) is an agent-oriented platform that uses the Jason framework to automate a UAV. The platform uses specific microcontrollers for aerial vehicles and it is generic for open-source microcontrollers. However the platform only permits integration with aerial vehicles.

The platform provides sixteen new Jason's internal actions to control the vehicle, including four internal actions to communication. The actions are responsible for the actuators/sensors controlling. But some of those actions are specific for aerial vehicles. The *RxTxComm* library is used to send/receive information from the real environment to the agent.

In this paper, the vehicle's platform allows the programmer to implements his/her own agent's behaviours based on the analysis of the electrical signals of the chosen vehicle. The implementation can be done for any kind of vehicles. Besides, it is possible to use any microcontroller, together with an OS, to control the vehicles.

The vehicle's platform is more flexible about all the layers integration. However it is not the simplest way of hardware automation, requiring some electrical skills from the programmer.

6 CONCLUSIONS

This paper presented an agent-oriented ground

vehicle that uses an unmanned vehicle platform that integrates a series of layers from the hardware connections layer until the agent reasoning layer programmed using the Jason framework. The platform consists of five layers that have to be programmed in a four-step methodology.

The paper also presented an example of a simple vehicle automation. The vehicle was embedded with an ATMEGA328 microcontroller where the motor's functions were programmed too. Afterwards, the serial interface methods were programmed along with the agent's action methods in order to represent the modifications that an agent can perform in the real world. Finally, the agent reasoning is programmed using the Jason framework, an agent-orient framework that uses the BDI approach.

The platform allows the automation of all types of vehicles and can work with all kind of hardware. The major objective of the platform is to provide a simple agent-oriented methodology that can be used for unmanned vehicles for any programmer.

The layers are composed of extent technologies that are widely used by programmers. In fact, there are no difficulties to handle those codifications steps. The methodology forces the programmer intervention in all development phases, providing a certain degree of freedom in the components selection. However, the platform demands much more development time and movements expertise compared with the other platforms.

For future works, a set of Jason's internal actions using latitude and longitude to provide global movements, based on the GPS values, will be developed instead of the global movements class developed to be executed in the simulated environment. It will allow the reasoning agent to control all the cognitive reasoning about global movements. The reasoning agent will not have to wait the end of the simulated environment processing cycle because the global position functions will be internally processed into agent's mind.

It will be necessary corrections in the movement's equation for long distances because of the Earth's eccentricity. The magnetic field can also interferes if the vehicles were embedded with a electronic compass. So, alternative solutions for orientation may be developed.

The external communication between unmanned vehicles for Multi-agent systems development will be implemented too. The Jason framework provides a group of communication internal actions that works along only with virtual agents into the same simulated environment. However this communication does not work between vehicle's that not share the same simulated environment. So, a group of firmware communications functions that receives the data from another vehicle into the real world and transfer it properly until reach the Reasoning Agent layer are necessary.

ACKNOWLEDGEMENTS

We would like to acknowledge all the support of DIREX and DEAC division at CEFET/RJ.

REFERENCES

Bellifemine, F., Caire, G., and Greenwood, D. (2007). *Developing multi-agent systems with JADE*. Wiley series in agent technology.

Bordini, R. H., Hubner, J. F., and Wooldridge, W. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. Jonh Wiley and Sons, London.

Bratman, M. (1987). *Intention, Plans and Practical Reasoning*. Center for the Study of Languages and Information, Harvard University Press.

Busetta, P. (1999). JACK Intelligent Agents – Components for Intelligent Agents in Java. *AgentLink Newsletter*, Melbourne, Australia, v.2, pages 2-5.

Ferber, J. (1998). *Multi-agent systems: An Introduction To Distributed Artificial Intelligence.* Addison-Wesley, United Kingdom, London.

Hama, M. T., Allgayer, S. R., Pereira, E. C., and Bordini, R. H. (2011). UAVAS: AgentSpeak Agents for Unmanned Aerial Vehicles. In Proceedings of the *2nd Workshop on Autonomous Software Systems*.

Huff, N., Kamel, A., Nygard, K. (2003). An Agent Based Framework for Modeling UAV's. In Proceedings of *Computer Applications in Industry and Engineering,* Las Vegas, Nevada, USA, pages 139-144.

Karim, S., Heinze, C. (2005). Experiences with the Design and Implementation of an Agent Based Autonomous UAV Controller. In Proceedings of *Autonomous Agents and Multiagents Systems*, Melbourne, pages 19-26.

Milone, E. F., and Wilson, W. J. F. (2008). *Solar System Astrophysics: Planetary Atmospheres and the Outer Solar System*. Springer, New York.

Wallis, P., Ronnquist, R., and Lucas, A. (2002). The Automated Wingman – Using JACK intelligent agents for Unmanned Autonomous Vehicles. In Proceedings of *Aerospace Conference*, *IEEE*, volume 5, pages 2615-2622.

Wooldridge, M. (2000). *Reasoning about rational agents*. Intelligent robotic and autonomous agents. MIT Press.