Distributed UAV-Swarm Control Using BDI Agents*

Bruno de Lima[1], Iago Silvestre[1], Pedro Henrique Dias[1], Leandro Buss Becker[1], Jomi Fred Hübner[1], Maiquel de Brito[2]

[1] Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

[2]Departamento de Engenharia de Controle, Automação e Computação
Universidade Federal de Santa Catarina (UFSC)
Blumenau – SC – Brazil

{bruno.szdl,iagosilvestre2004,pedrohenrique.dias8}@gmail.com,

{leandro.becker, jomi.hubner, maiquel.b}@ufsc.br

Abstract. *This paper describes the development of a swarm of autonomous unmanned aerial vehicles (UAVs) that collaborate with each other in forests fire-fighting. These vehicles are controlled by BDI agents. The solution is distributed and decentralized. Each agent, embedded in a UAV, uses its knowledge and resources to solve a broader problem.*

1. Introduction

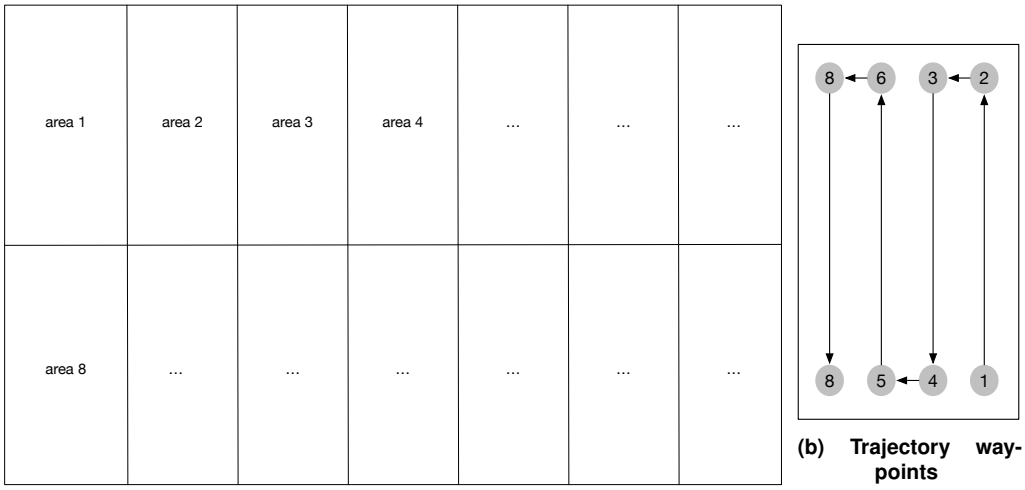
In recent years, Unmanned Aerial Vehicles (UAVs) have been applied to many fields, ranging from military to commercial and educational applications [Fahlstrom and Gleason 2012]. Some of these applications require decentralized solutions with distributed processing by autonomous, cooperative UAVs. These features are, to some extent, provided by agents and Multi-Agent Systems (MAS). This paper describes the use of MAS in a team of UAVs that fight fires in a scenario with the aforementioned requirements. The UAVs simulated with Gazebo [Koenig and Howard 2004]. Each UAV is controlled by an agent. The agents cooperate with each other to find and extinguish fire spots.

2. Purpose of the application

The application described in this paper regards multiple UAVs working on fire detection and fighting in forests. It requires autonomy of each UAV, decentralization of decision and action, distribution of the decision process and resources usage, and effective cooperation. These features provided by agents and Multi-Agent Systems [Boissier et al. 2020]. Thus, the proposed solution is based on BDI agents, that are programmed in terms of Beliefs, Desires and Intentions. Beliefs represent information about the environment (given by sensors) and other agents (given by communication). Desires are potential objectives of the agent (give by the developer or other agents, in our case). Intention are desires the agent is actually pursuing and has a plan of action selected for that.

In this application, BDI agents have physical bodies since some beliefs come from physical sensors, as well as some of the required actions are enabled by physical actuators.

*This short paper describes a demonstration submitted to WESAAC.



(a) Areas of the scenario

(b) Trajectory waypoints

Figure 1. Areas of the scenario and waypoints of the trajectory to find fire in each area

Thus, building the solution involves (i) using BDI concepts to develop a distributed strategy for firefighting and (ii) integrating the perception and action processes of the agents with the sensors and actuators that compose the UAVs.

3. Demonstration

The development of the application involves the following points, described in the next sections: (i) developing the distributed strategy and (ii) implementing it with BDI agents.

3.1. Team Strategy

Each UAV is controlled by a BDI agent that has information about the state of the environment (the forest, trees, fire spots, other agents, etc.) and about the application (e.g., plans to follow a trajectory, join other agents, wandering). The agents coordinate with each other to perform the following tasks:

— **Task 1: Finding fire spots.** The area is divided among the agents and each agent is responsible to verify the existence of fire there. For n agents, the scenario is divided in n equally sized areas as shown in Fig. 1a. Inside each area, the corresponding agent flies following a trajectory that covers all its area. In the example of Fig. 1b, 8 waypoints are required for the area to be covered. The distance between the vertical arrows is defined by the range (on the ground) of the fire sensor. It is thus impacted by the altitude flight of the UAV. When the fire sensor detects an occurrence of fire, the agent acts to extinguish the fire. If the last waypoint is achieved, it finishes its mission.

— **Task 2: Extinguish fire.** When one agent finds the first spot of fire, it asks other agents to come help him to extinguish that fire. We are assuming that usually a single UAV is not capable to extinguish some fire and m agents are required for that task. m is computed based on the size of the area with fire and the capacity of extinguishing fire of each UAV. The agent who found the fire spot, broadcasts the finding (i.e., the fire location) to other agents asking for help. The $m-1$ nearest UAVs answer the request. The agents can then work on the fire. After extinguishing the fire, all involved agents resume looking for fire spots. To elect the $m-1$ agents, a decentralized protocol as follow is conceived:

1. the agent i who found the fire broadcasts a "fire found" message to others;
2. all agents who received the message answer with their current location if (i) they are executing task 1 and (ii) are not participating in other protocol like this; otherwise, they answer "not available";
3. the agent i waits answers for some time or until $n - 1$ answers were received;
4. based on its own location and locations received from others, agent i computes the $m - 1$ nearest UAVs, we named these UAVs "helpers";
5. agent i sends a message to helpers asking them to come;
6. agent i sends a message to others (non helpers) that they are dismissed.

— **Task 3: Finishing the mission.** When the agent has verified the entire area and extinguished all the fire, it broadcasts a message informing others that it has finished and flies to its landing location.

3.2. Design and implementation of the agents

The implemented agents extend the default Jason agents [Bordini et al. 2007].[1] Such extension is provided by the *embedded-mas* framework.[2] This is a general purpose framework to implement software agents embodied in physical devices which, in this project, are the UAVs simulated with Gazebo. The agent interacts with the external environment through the processes of perception and action. The extended Jason agents include two main features related to these processes: (i) values acquired by sensors are converted into perceptions of the agent, which are taken into account in reasoning process; and (ii) actions enabled by the physical actuators are included in the repertory of the actions of the agents.

The body of each agent is a Turnigy SK450 quad copter equipped with a 5MP camera to detect images of the environment. Fire is detected by processing these images. These elements are simulated using Gazebo integrated with the Robot Operating System (ROS) [Koubaa 2017]. ROS resources (i.e. topics and services) are integrated to the processes of perception and action of the agents. Topic values are converted to perceptions. The actions of the agents are converted either in topic writings or in service calls, depending on the application requirements. This is a seamless integration that preserves the agent-oriented programming style. The agents are programmed through the usual high-level constructs such as beliefs, goals, and plans. In runtime, beliefs are updated based on the perceptions of the agents while goals are satisfied through the execution of plans that consider the beliefs of the agents and usually require the agents to execute some actions. The perception and action process are integrated with ROS in the underlying implementations of the agent execution machinery.

Fig. 2 shows an excerpt of the code of the agent. It implements part of the protocol to ask other agents to come and help to extinguish some fire. It shows the main BDI agent constructs integrated with the hardware to implement the agent UAVs. For example, `current_position(CX, CY, CZ)` is a belief coming from the UAV sensor values, `!extinguish_fire(CX, CY)` is a desire, `+detected_fire` is an event, and `defaultEmbeddedInternalAction` is an internal action enabled by the actuators present in the hardware.[3]

[1] `http://jason.sf.net`

[2] `http://github.com/embedded-mas/embedded-mas`

[3] The full code is available at `https://github.com/iagosilvestre/start-UFSC`.

```
+detected_fire                    // whenever a fire spot is detected by me
   :  current_position(CX, CY, CZ)        // and I am in location Cx,Cy,Cz
      & .intend(follow_trajectory(CW))  // and I am currently finding fire
   <- .suspend(follow_trajectory(CW));      // stops the finding intention
      .broadcast(tell, found_fire(N));                    // notify others
      !elect_helpers(CX,CY);                // select and wait for helpers
      !extinguish_fire(CX, CY);         // add new goal to extinguish fire
      .resume(follow_trajectory(CW)).            // return to finding fire

+!help(N,X,Y)          // when I was elect to help agent N for fire at X,Y
   <- .suspend(follow_trajectory(CW));
      !goto_fire(X, Y);                 // I desire to go to the fire spot
      !extinguish_fire(X, Y);           // I desire to extinguish the fire
      .resume(follow_trajectory(CW)).            // return to finding fire

+!goto_fire(X, Y, Z)                // when I desire to go to the fire spot
   <- !check_near(X, Y, Z).  // continuously check whether I am at the spot

+!check_near(X, Y, Z) :  near(X, Y)       // I am arrived to the fire spot
   <- .print("Arrived at ", S).

+!check_near(X, Y, Z)                       // I am going to the fire spot
   <- //execute an internal action to take the UAV to (X,Y,Z)
      defaultEmbeddedInternalAction("roscore1","goto", [X, Y, Z]);
      !check_near(X, Y, Z, S).


+!elect_helpers(CX,CY)    <- ...                  // plan to elect helpers
+!extinguish_fire(CX, CY) <- ...      // plab to extinguish fire at Cx, Cy
```

Figure 2. Code excerpt

4. Conclusion

The described implementation has been provided a solution with the following main features: autonomy of the UAVs, hardware decoupling, decentralization of the solution and cooperative behaviour of the UAVs. Future work include moving from simulation to real hardware and improvements in the team strategy.

References

Boissier, O., Bordini, R. H., Hübner, J., and Ricci, A. (2020). *Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo*. MIT Press.

Bordini, R. H., Hübner, J. F., and Wooldrige, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons.

Fahlstrom, P. G. and Gleason, T. J. (2012). *Introduction to UAV Systems*. John Wiley & Sons.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2149–2154.

Koubaa, A. (2017). *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Springer, 1st edition.