Martin MOLINA, Alberto CAMPORREDONDO, Hriday BAVLE, Alejandro RODRIGUEZ-RAMOS, Pascual CAMPOY, 2019. An execution control method for the Aerostack aerial robotics framework. *Frontiers of Information Technology & Electronic Engineering*, 20(1):60-75.
https://doi.org/10.1631/FITEE.1800552

An execution control method for the Aerostack aerial robotics framework

Key words: Aerial robotics; Control architecture; Behavior-based control; Executive system

Corresponding author: Martin MOLINA
E-mail: martin.molina@upm.es
ORCID: http://orcid.org/0000-0001-7145-1974

Motivation

1. Execution control is a common task in robot control architectures to communicate two description levels:

   (1) A level where a requester tells a robot what to do using symbolic commands;

   (2) A level where a number of computational processes concurrently run to generate the required functionalities.

2. The methods used for execution control have a deep impact on the quality of the final system, such as the safety and complexity of human-robot interaction.

3. There is a need for methods for building system architectures emphasizing utility aspects related to timeliness in development and performance.

Context

The method for execution control is a part of the Aerostack framework.

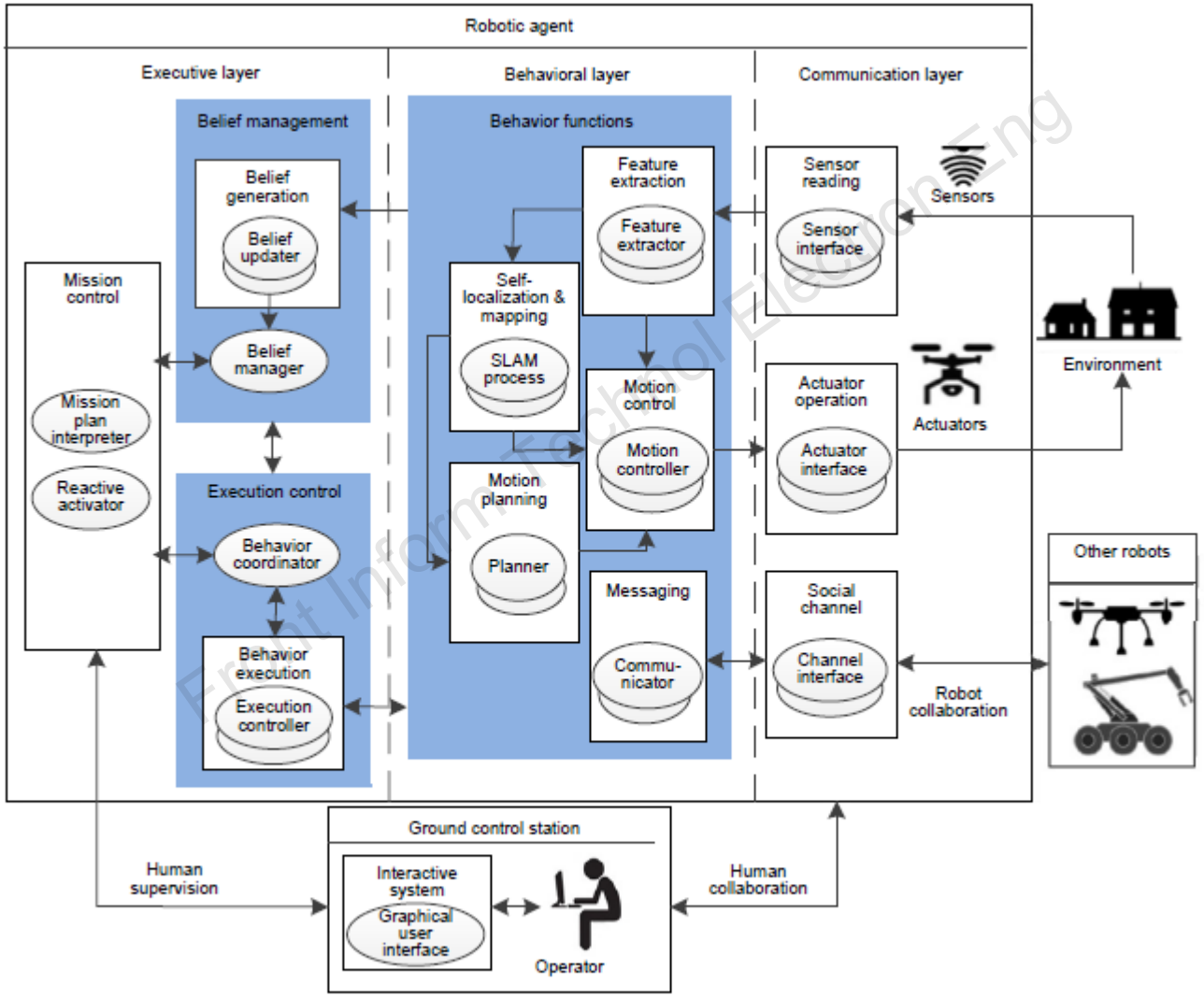

Fig. 1 Aerostack architecture (version 3.0)

Challenges

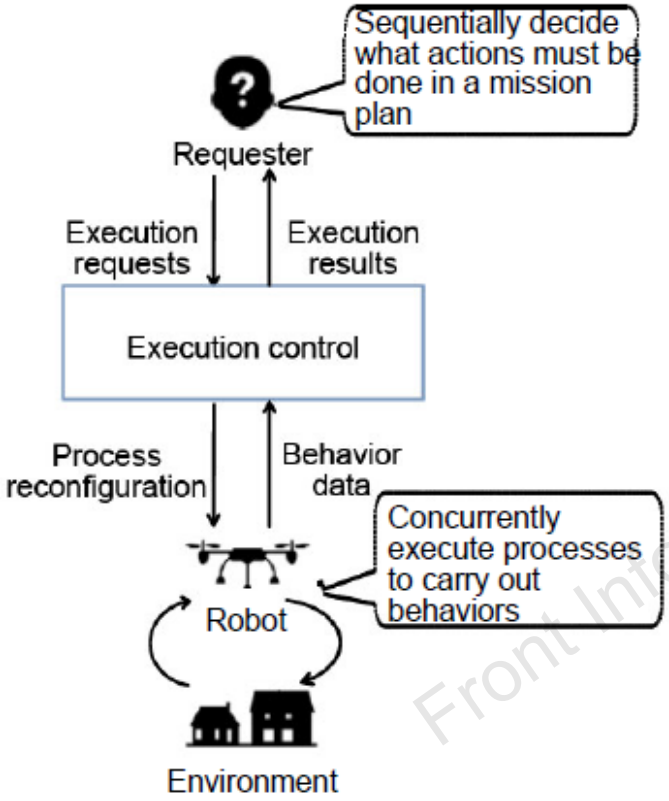

Fig. 2 Role of the execution control system

1. Appropriate representation for the requester at an adequate level of abstraction;
2. General. Applicable in the construction of different aerial robotic systems and missions;
3. Usable. Useful for a developer in building the execution control system of robot architectures;
4. Use of standards. Using programming standards of the robotic systems that are used by Aerostack (e.g., ROS middleware);
5. Accessible. Freely accessible as a part of an open-source framework.

Method (1/7)

Representation used by the execution requester:

Behavior activation requests

TAKE_OFF
GO_TO_POINT coordinates: (1.0, 2.5, 7.0)
ROTATE angle: 45

Memory of beliefs

Cause of behavior termination:

1. Goal achieved;
2. Time out;
3. Wrong progress;
4. Interrupted.

Table 1 Examples of predicates representing beliefs

Predicate	Description
Object(x, y)	Object x is an instance of class y
Position(x, y)	Object x is at the position y
Name(x, y)	Name of object x is y
Flight_state(x, y)	Aerial robot x is in flight state y
Code(x, y)	Numerical code of x is y
Color(x, y)	Color of x is y
Charge(x, y)	Charge of x is y
Carry(x, y)	Agent x carries object y
Visible(x)	Object x can be observed

Method (2/7)

Central coordination

Behavior-based distribution

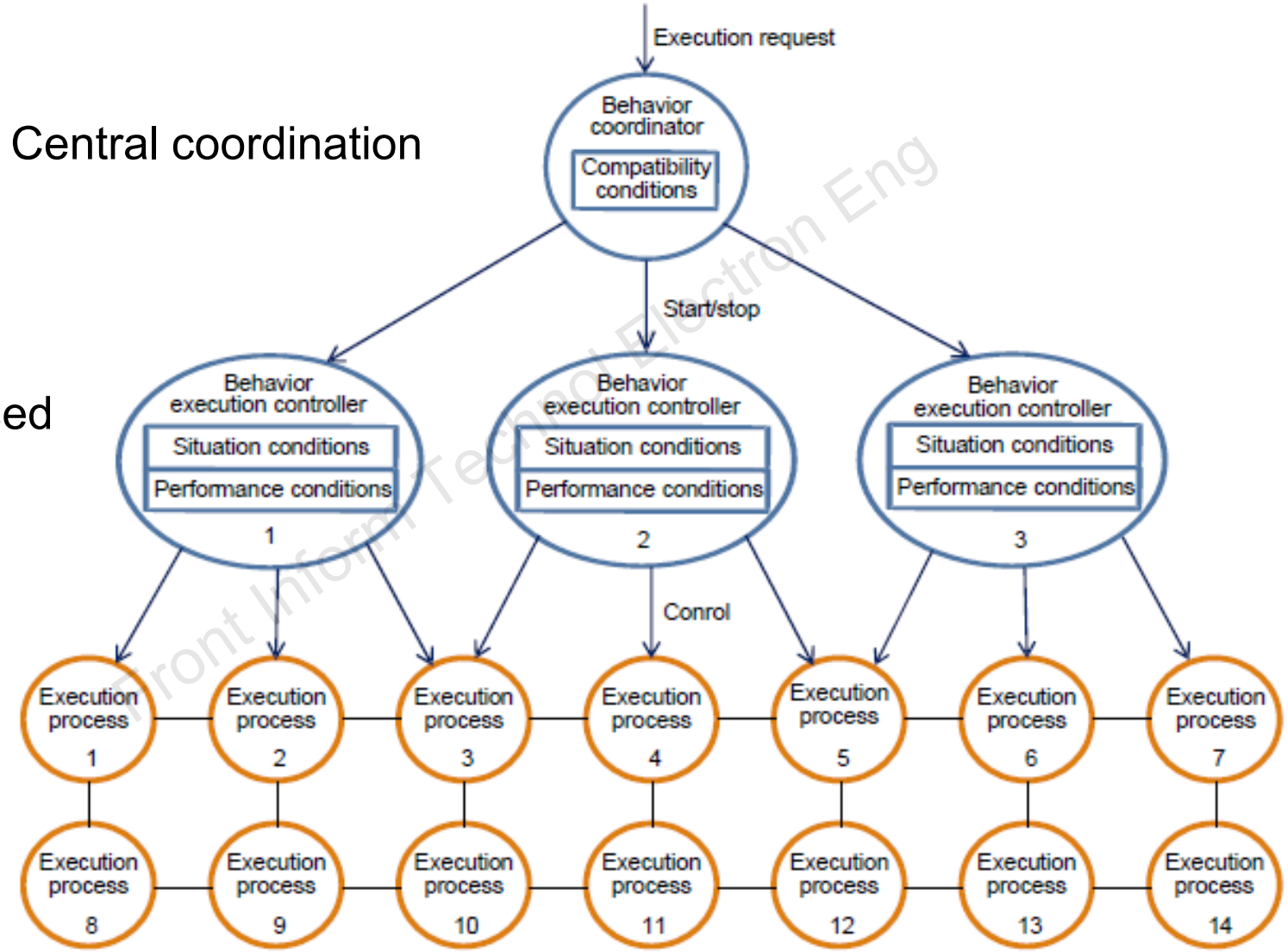

Fig. 3 Distributed scheme for execution control

Method (3/7)

Functions of the behavior execution controller:

1. [Before the execution]

Checks if the behavior can be activated. Property: “situated behaviors”

Situation conditions:

- Behavior: TAKE_OFF Condition: the robot is landed
- Behavior: GO_TO_POINT Condition: the robot is flying

2. [At the beginning of the execution]

Launches necessary processes and publishing messages.

3. [During the execution]

Monitors the execution. Property: “cognizant failure”

•Performance condition:

- Behavior: TAKE_OFF Condition: timeout 10 s
- Behavior: GO_TO_POINT Condition: distance to destination decreases

4. [At the end of the execution]

Informs about the cause of termination.

Method (4/7)

Functions of the behavior coordinator:

1. Accept or reject a request (activation/inhibition):

   Compatibility constraints and priority scheme;

2. Deactivate other incompatible behaviors:

   Compatibility constraints;

3. Activate other necessary behaviors:

   Default behaviors.

Method (5/7)

Detailed architecture of the execution control system:

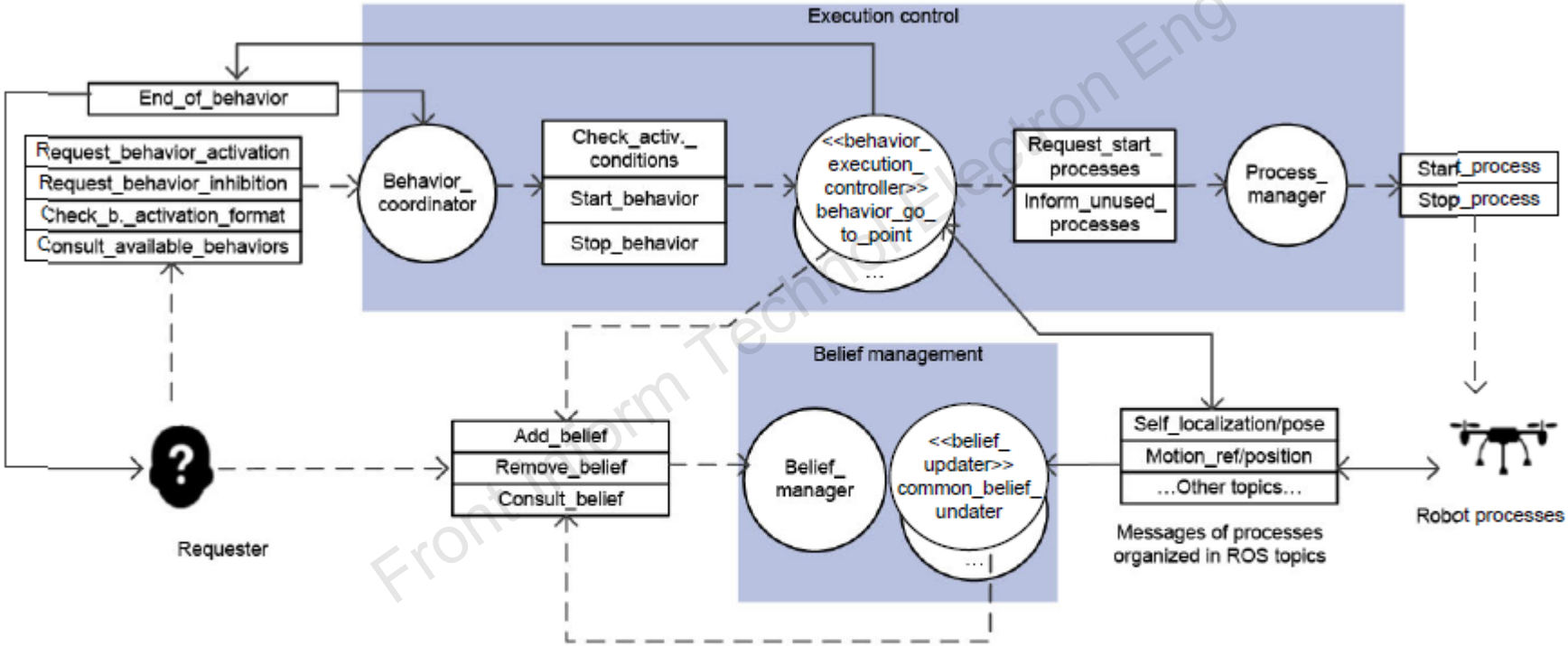

Fig. 4 Detailed architecture of the execution control system

Circles: Aerostack processes (ROS nodes); rectangles with dashed arrows: request-reply ROS services; rectangles with continuous arrows: publish-subscribe messages as ROS topics

Method (6/7)

Catalog of behaviors with compatibility conditions:

File: behavior_catalog.yml

```
behavior_descriptors:

- behavior: GO_TO_POINT
    category: goal_based
    timeout: 120
    default: no
    processes:
      - droneTrajectoryController
      - droneTrajectoryPlanner
      - droneYawPlanner
    parameters:
      - parameter: coordinates
        allowed_values: [-100,100]
        dimensions: 3
...
```

```
compatibility_conditions:

- mutually_exclusive:
  - TAKE_OFF
  - LAND
  - KEEP_HOVERING
  - GO_TO_POINT
  - ROTATE
  ...

- active:
    - SELF_LOCALIZE_BY_VISUAL_MARKERS
  before:
    - GO_TO_POINT
    - ROTATE
    - KEEP_HOVERING
    ...
```

Method (7/7)

Example of behavior coordination:

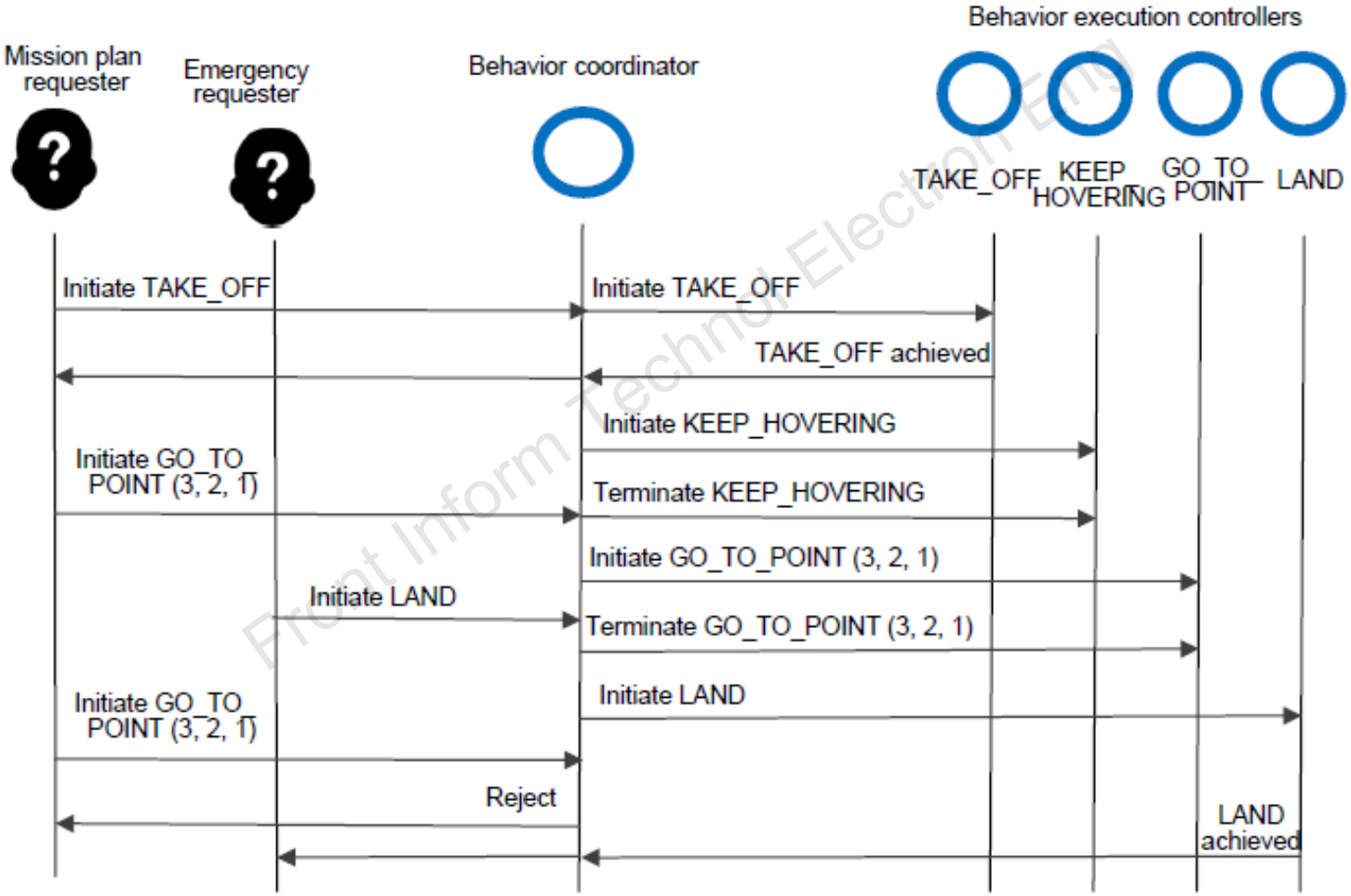

Fig. 6 Example of behavior coordination

Major results (1/5)

Simulated flights using RotorS

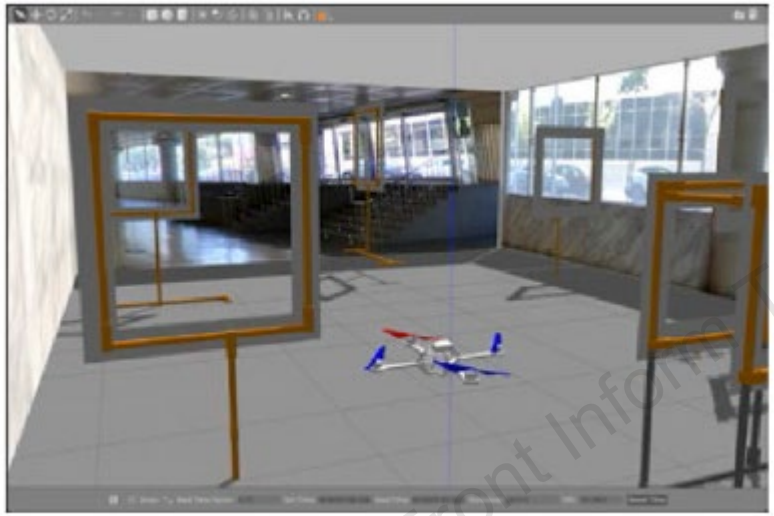

Fig. 8 Flight simulation using RotorS

Real flights using Parrot Bebop 2

Fig. 9 A real flight experiment

Laptop computer: CPU Intel i7-7700HQ, 8 cores, 2.8 GHz, 16 GB RAM

Major results (2/5)

Example trajectory generated in a real flight:

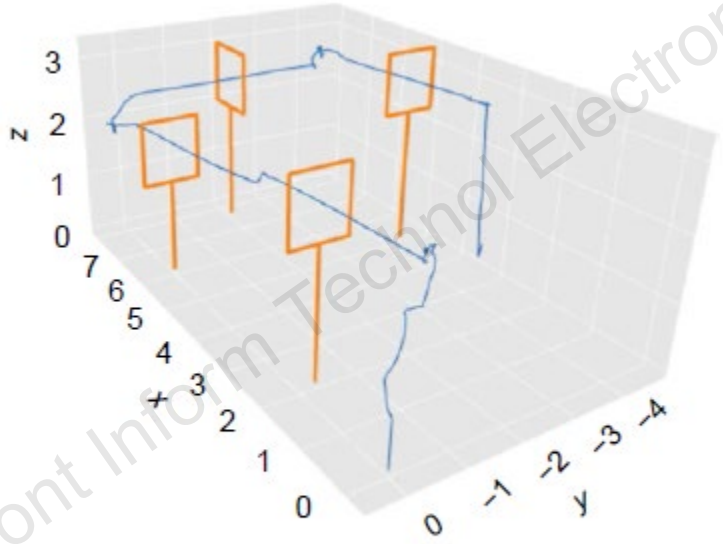

Fig. 11 An example of the trajectory generated in a real flight

Trajectory completed in 1 min 58 s

Major results (3/5)

Details of the trajectory to cross the first frame:

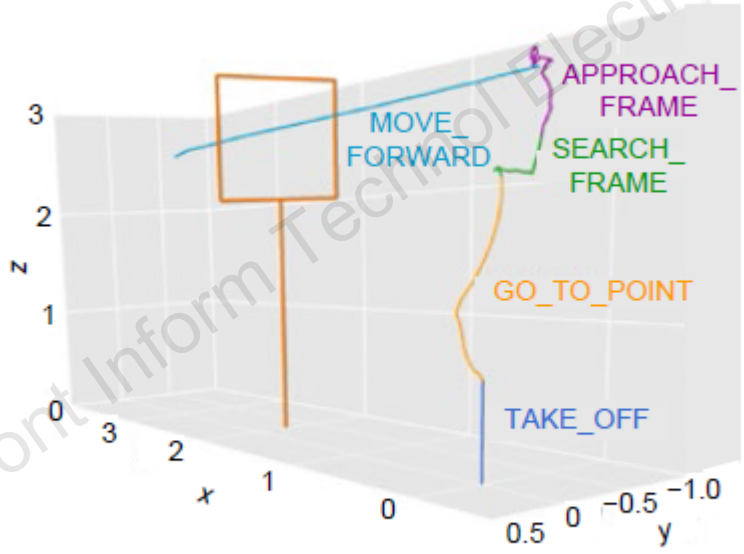

Fig. 12 Details of the trajectory to cross the first frame

Major results (4/5)

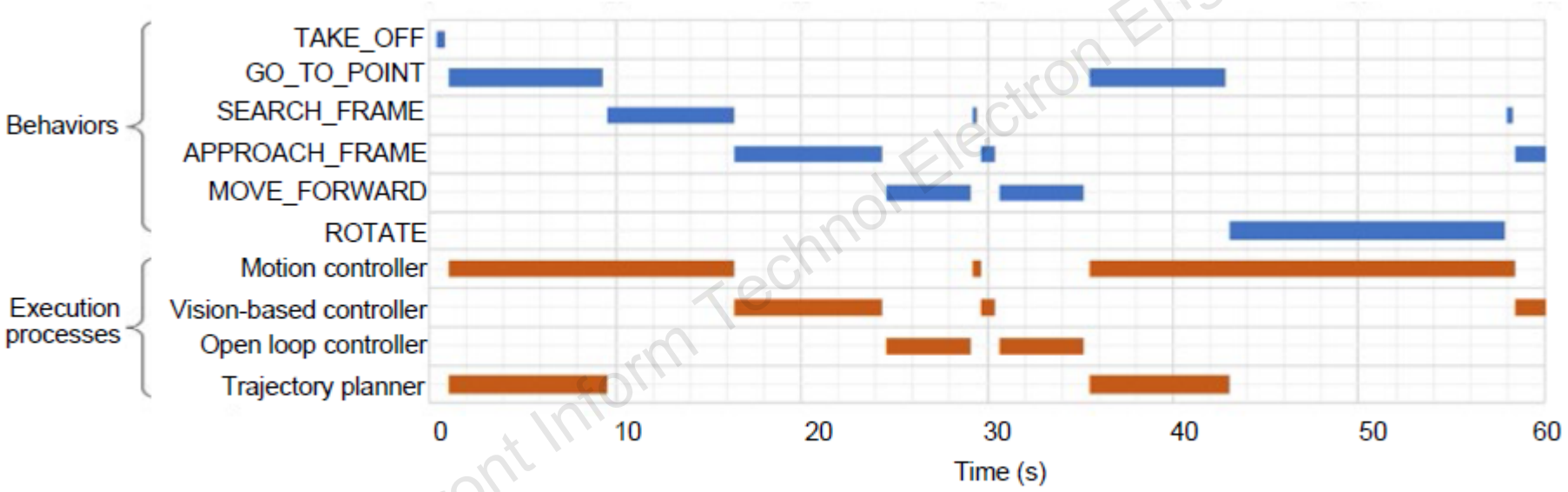

Fig. 10 Partial sequence of behavior activations (blue) and execution of processes (red)

Behavior coordination time: 119 ms (average), 36 ms (minimum), 204 ms (maximum)

Major results (5/5)

Table 2 Evaluation numbers of programming effort in person-hours

Behavior	Evaluation number (person-hour)				
	L	D	P	V	Total
GO_TO_POINT	16	24	16	16	72
MOVE_FORWARD	0.5	1.0	5.0	3.0	9.5
SEARCH_FRAME	1	2	5	10	18
SELF_LOCALIZATION (M1)	1	1	8	4	14
SELF_LOCALIZATION (M2)	56	16	24	16	112

L: learning the behavior implementation; D: designing the execution controller; P: programming the execution controller; V: validating the integration

Code size	Number of lines
Execution control system (common part)	4 754 lines
Execution control of a behavior (average)	368 lines

Conclusions

1. The method uses an original design:
   (1) Distributed organization based on behaviors;
   (2) Central coordination with a catalog with compatibility conditions.
2. The method is:
   (1) Integrated in the Aerostack framework (version 3.0);
   (2) Open-source with license BSD;
   (3) Implemented with standards used in robotics (e.g., ROS).
3. The experimental tests demonstrate that the method is valid for practical applications in aerial robotics and can be applied to specific systems with acceptable development effort.

Acknowledgements

1. The authors are members of the research group CVAR (Computer Vision and Aerial Robotics) of the Universidad Politecnica de Madrid, Spain (www.visionaerialrobotics.com).

2. This work has received funding from the European Union's Horizon 2020 Research and Innovation Program under the project ROSIN (No. 732287).