Specifying Complex Missions for Aerial Robotics in Dynamic Environments

Martin Molina[1]*, Adrian Diaz-Moreno[1], David Palacios[1], Ramon A. Suarez-Fernandez[2], Jose Luis Sanchez-Lopez[2], Carlos Sampedro[2], Hriday Bavle[2] and Pascual Campoy[2]
[1]Department of Artificial Intelligence, Technical University of Madrid, UPM, Spain
[2]Computer Vision Group, Centre for Automation and Robotics, CSIC-UPM, Spain

Abstract

To specify missions in aerial robotics, many existing applications follow an approach based on a list of waypoints, which has been proven to be useful and practical in professional domains (agriculture of precision, creation of terrain maps, etc.). However this approach has limitations to be used in other problems such as the one defined in the IMAV 2016 competition (e.g., a search and rescue mission). In this paper we describe a language to specify missions for aerial robotics that tries to overcome such limitations. This language has been designed as part of a complete software framework and architecture for aerial robotics called Aerostack. The paper describes the result of experimental evaluation in real flight and its adequacy for the IMAV 2016 competition.

1 Introduction

There are a number of available software applications that allow operators to specify a mission for aerial vehicles using a ground control station. Examples of such applications are [1]: MP (Mission Planner), APM Planner 2, MAVProxy, QgroundControl, and PC Ground Station (DJI). However, the use of waypoint lists to specify a mission presents important limitations [1, 2]. One of the most significant limitations is that they are rigid descriptions that are not able to adapt to mission circumstances. The specification is normally based on a fixed list of waypoints that cannot change dynamically (e.g., in the presence of dynamic obstacles). The specification follows a main sequential workflow and it is difficult to specify alternative flows (e.g., iterations, branches, conditional waypoints, alternative flows).

In order to cope with these limitations, mission specification languages can follow more powerful planning approaches. For example, there are classical planning approaches in artificial intelligence (e.g., with STRIPS-like operators). However, to be used successfully in robotics, they need to be adapted or combined with other solutions to react efficiently to the environment changes, monitor mission execution and integrate with reactive behaviors [3]. Some practical planning solutions in robotics are task-based specifications, Petri nets, or behavior-based specifications besides others (rule-based reactive planners or finite state machines)

In this paper, we present the results of our recent research work to create a language to specify complex missions for aerial robotics, such as the IMAV 2016 indoor competition. The goal of this mission is a search and rescue task with different sub-missions related to search and map an unknown environment and transport objects to certain locations.

Our proposed solution has been inspired by the task-based approach for planning but includes also a particular definition of basic concepts (e.g., actions and skills) and it is combined with an event-driven reactive planning approach, to facilitate adaptation to changes of the environment. The solution has been integrated as part of the software framework Aerostack (www.aerostack.org).

The remainder of the paper is structured as follows. Section 2 describes the characteristics of our proposed language. Section 3 describes how to verify the specification. Section 4 presents how our planning system was implemented in the software framework Aerostack. Finally, Section 5 describes the results of the experimental evaluation based on real flights and its ability to specify a mission for the IMAV 2016 competition.

2 Mission specification in TML language

We designed the language called TML (Task-based Mission specification Language) using XML syntax to be readable by both humans and machines. TML provides a logical level using more natural and intuitive conceptual notions that avoids the need of specifying technical details at programming level (e.g., what specific processes must be executed for each goal or in which order they must be executed).

Example 1 shows a specification of a simple mission in TML language. In this language, the mission is defined with the tag <mission> and the attribute name and the body of the mission is separated in two main parts: task tree and event handlers.

*Email address: martin.molina@upm.es

[1]Mission Planner http://planner.ardupilot.org/planner/index.html,
APM Planner 2.0: http://planner.ardupilot.org/planner2/index.html,
MAVProxy: http://dronecode.github.io/MAVProxy/html/index.html,
QgroundControl: http://qgroundcontrol.io,
PC Ground Station: http://www.dji.com/es/product/pc-ground-station

2.1 *Task tree*

TML uses the notion of *task* as a basic component to structure a mission with a modular organization. For example, in the case of the IMAV 2016 competition, tasks can identify the different parts of the global mission such as take off from a moving platform, entering the building, pickup objects, etc. The mission is specified with a set of tasks organized in a task tree. Each task is specified using the tag <task> and the attribute {name}.

```
<mission name="Mission example">
  <task name="Initial take off">
    <action name="TAKE_OFF"/>
  </task>
  <task name="Navigate avoiding obstacles">
    <skill name="RECOGNIZE_VISUAL_MARKERS"/>
    <skill name="AVOID_OBSTACLES"/>
    <task name="Stabilize at origin">
      <action name="STABILIZE"/>
    </task>
    <task name="Go to destination">
      <action name="GO_TO_POINT">
        <argument name="point"
                  value="(3.0,4.0,1.5)"/>
      </action>
    </task>
  </task>
  <task name="Final landing">
    <action name="LAND"/>
  </task>
  <event_handling>
    <event name="Land command recognized">
      <condition parameter="visualMarker"
                 comparison="equal" value="3"/>
      <action name="LAND"/>
      <termination mode="ABORT_MISSION"/>
    </event>
  </event_handling>
</mission>
```

Example 1: Mission specification in TML language.

We use the concept of *action* to express an elementary goal that the aerial robot is able to achieve by itself, using its own actuators. An illustrative set of actions related to IMAV 2016 competition is the following: take off, go to a point, move forwards, explore space, pick up item, drop item, rotate yaw, and land.

To represent a particular robot's ability we use the concept of *skill*. An illustrative set of skills related to IMAV 2016 competition is the following: avoid obstacles, self locate by markers, recognize buckets and recognize items. Skills can be active or inactive in a particular robot. In general, skills have influence in the behavior of actions. Thus, we can understand skills as global modifiers for sets of actions.

The notion of skill is useful as an intuitive concept to help operators express more easily what complex abilities should be active, without considering low-level technical details. Internally, a skill is automatically supported by a set of running processes. Thus, the activation of skills is associated to the increase of resource consumption (memory space, processing time, battery charge) so it is important to deactivate unnecessary skills when it is possible.

In TML, the body of a terminal task (i.e., a task that is a terminal node in the task tree) must specify an action with the tag <action> and the attribute {name}. An action can include optionally arguments. Arguments use the tag <argument> and the attributes {name, value}. The body of a terminal task can also specify several skills to be active with the tag <skill> and the attribute {name}.

Action	Description
FLIP	The vehicle performs a flip in a certain direction (argument direction with the values {front, back, right, left}). The direction by default is to the front.
GO_ TO_ POINT	The vehicle moves to a given point. The point can be expressed using absolute coordinates (argument point) or coordinates relative to the vehicle (argument relative point).
LAND	The vehicle descends vertically (through the z axis) until it touches the ground. It is assumed that the ground is static.
ROTATE_ YAW	The vehicle rotates the yaw a number of degrees (argument angle).
STABILIZE	The vehicle tries to cancel all the perturbations and turbulences that may affect its system such as movement speeds and attitude speeds.
TAKE_ OFF	The vehicle takes off from its current location to the default altitude. If the vehicle is flying, this action is ignored.
WAIT	The vehicle waits on the air for a specified number of seconds (argument time).

Table 1: List of actions (partial) used in TML language.

The body of a non-terminal task can include skills that are active during the task execution. The body of a non-terminal task does not include actions. Instead, the body includes one or several simpler tasks that are executed in sequence.

The linear sequence of task execution can be modified with repetitions and conditioned executions. The tag <repeat> and the attribute {times} is used to repeat a task several times. The tag <condition> and the attributes {parameter, comparison, value} are used to establish a condition to execute a task. The allowable values for the attribute comparison are {equal, less than, less than or equal to, greater than, greater than or equal to, not equal to}. Example 2 illustrates the use of these tags.

```
<task name="Flip and move if green">
  <task name="Flip three times">
    <repeat times="3"/>
    <action name="FLIP"/>
  </task>
  <task name="Move if green">
    <condition parameter="observed color"
               comparison="equal" value="green"/>
    <action name="GO_TO_POINT">
      <argument name="point"
                value="(3.0, 4.2, 1.5)"/>
    </action>
  </task>
</task>
```

Example 2: Task body with conditions in TML language.

2.2 *Event Handlers*

We have combined the task-based solution with reactive planning to provide more adaptability to changes of the environment. To represent conditions about the world state, we use the concept of event. An event is the occurrence of a significant change in the state of the external environment or the state of the own robot. Events can be related to normal situations (e.g., the detection of a specific visual marker) or abnormal/undesired situations (time out, discharged battery, etc.).

In TML, the mission specification includes a section for event handlers defined with the tag <event_ handling> to describe what to do in the presence of events. Event handlers are similar to condition-action rules, a typical representation used in reactive planners. Events are defined with the tag <event> and the attribute {name}. Each event includes a list of conditions (in conjunctive form), a list of actions to be done and an ending step defined with the tag <termination> and the attribute {mode}. Events can use a particular type of condition with the attribute {currentTask}. This is useful to express that the event happens while a particular task is being executing. Example 3 illustrates this case in TML language.

```
<event name="take off failure">
  <condition currentTask="initial take off"/>
  <condition parameter="action confirmation"
             comparison="equal" value="failure"/>
  <termination mode="ABORT_MISSION"/>
</event>
```

Example 3: Event declaration in TML language.

The task tree and the event handlers are integrated during the execution time in the following way:

1. The interpreter navigates in the task tree following a depth-first control strategy. When the interpreter reaches a terminal node of the tree, it requests the execution of the corresponding action with the active skills. The order of this control strategy can be modified by the user with repetition and condition sentences.

2. While the interpreter waits until a requested action finishes, it verifies the conditions of event handlers. If an event handler verifies its condition, it can request additional skills to be active, can request additional actions to be done or can change the normal execution flow (abort mission, abort task, jump to task, etc.).

3 Verification of the Mission Specification

The text in TML language written by the operator must be automatically analyzed to verify that it does not include errors generated by mistakes or misunderstood concepts. This is important to guarantee a robust operation avoiding system crashes. In general, the verification of a mission specification can include two types of analysis:

- Language validation. This corresponds to the lexical, syntax and semantic validation of the mission specification according to the grammar defined for the TML language.
- Physical feasibility verification. This verification checks if the specified mission can be performed in practice considering constraints of the physical world. This validates, for example, that a point to reach is not too close to an obstacle or that the distance to cover is not too long for the remaining battery.

The verification of physical feasibility is a complex task with different dimensions. It includes (1) the *individual verification* of the feasibility of each particular action, (2) the *contextual verification* to check the feasibility of an action or a skill in relation to other skills and actions that occur at the same time (3) the *temporal verification* of the complete mission taking into account the temporal evolution of the whole set of actions and skills.

The relative time when the verification is performed is another significant distinction. The verification can be done before the mission starts. For example, it is possible to verify in advance that a certain spatial point is too far to be reached, considering the maximum charge of battery. On the other hand, the verification can be done *during* the mission execution, considering unexpected changes in the environment such as the moving obstacles, bad illumination, etc.

We designed a solution to verify the physical feasibility using a computational model with a constraint-based representation. This model uses variables, parameters, functions and constraints. Variables $\{x_i\}$ represent the dynamic values of physical references and magnitudes (e.g., destination point, current charge of battery, etc.). Parameters $\{k_i\}$ represent constant values for physical magnitudes related to the performance of the robot such as maximum speed, battery consumption rate of the vehicle, etc. Parameters can be divided into vehicle-independent parameters (general for any kind of vehicle) or vehicle-specific for each category of vehicle. Functions represent spatio-temporal and motion functions (see Table 2) such as the length of a trajectory, distance to the closest obstacle, maximum distance covered with certain battery charge, required speed to reach a point at certain time, etc.

Function	Description
Distance(x_1, x_2)	Distance from point x_1 to point x_2
DistanceBattery(x_1, x_2)	Maximum distance covered with battery charge x_1 and consumption rate x_2
DistanceObstacle(x_1)	Distance from point x_1 to the closest obstacle
Length(x_1)	Length of trajectory x_1
Speed(x_1, x_2, x_3)	Required speed to departure now from point x_1 and arrive at point x_2 at time x_3
Trajectory(x_1, x_2)	Trajectory from point x_1 to point x_2 (generated by a trajectory planner)

Table 2: Example functions used in the verification model.

Constraints $\{c_i\}$ are conditions about the physical world that must be satisfied. Examples of these conditions are: the destination point must be safe from obstacles, there must be enough battery for the movement, and the destination point must be reachable at an acceptable speed. The previous conditions are represented with the following three constraints (with functions, variables and parameters):

c_1: $\mathit{DistanceObstacle}(x_2) > k_2$

c_2: *Length(Trajectory*$(x_1, x_2)) <$ *DistanceBattery*(x_3, k_1)
c_3: *Speed*$(x_1, x_2, x_4) < k_3$

where the variable x_1 is the current point, x_2 is the destination point, x_3 is the current battery charge, and x_4 is the planned time of arrival to the destination; and the parameter k_1 is the battery consumption rate, k_2 is the minimum free acceptable space between obstacle and vehicle, and k_3 is the maximum speed of the vehicle.

The whole set of constraints is divided in subsets according to the categories of actions. For example, there is a subset of constraints for actions related to rotation motions, another subset of constraints for actions related to translation motions, etc. For a given action, the verification procedure reviews only the subsets of constraints that correspond to categories to which the given action belongs.

This type of model is generic to be reusable for different physical platforms. Only the vehicle-specific parameter values must be manually configured.

4 THE TML INTERPRETER

We implemented an interpreter of the TML language with a set of computational processes in the software framework Aerostack. This section summarizes Aerostack and the set of processes that support the interpretation of the TML language.

4.1 *The Aerostack software framework*

Aerostack [4] is a software framework for aerial robotics, with a multilayered architecture supported by a distributed inter-process communication. Aerostack uses an architecture based on an extension of the hybrid reactive/deliberative paradigm, i.e., an architecture that integrates both a deliberative and reactive approaches [5]. The architecture includes five layers (Figure 1):

- The reactive layer with low-level control with sensor-action loops.
- The executive layer that accepts symbolic actions from the deliberative layer and generates detailed behavior sequences for the reactive layer. This layer also integrates the sensor information into an internal state representation.
- The deliberative layer generates global solutions to complex tasks using planning.
- The reflective layer helps to see if the vehicle is actually making progress to its goal and helps to react in the presence of problems (unexpected obstacles, faults, etc.) with recovery actions.
- The social layer includes communication abilities, which is important to establish an adequate communication with human operators and other robots.

The first three layers correspond to the popular hybrid design known as the three-layer architecture [6]. The reflective layer is based on cognitive architectures [7, 8] to simulate certain self-awareness able to supervise the other layers. The social layer has been proposed in multiagent systems and other architectures with social coordination (e.g., [9]). It is worth to highlight that the higher level layers, reflective and deliberative, must work slower (0.1 - 10 Hz.) than the lower level layers, executive and reactive (10 - 1000 Hz.).

4.2 *Processes for mission planning*

To implement an interpreter for our approach for mission specification, we have upgraded Aerostack, reusing some of the available components, and programming and integrating some additional ones. The processes that support the planning system perform the following tasks asynchronously, generating outputs when they receive new inputs that requires an action to be taken (Figure 2):

- The *mission planner* interprets the TML specification written by the human operator. The mission planner uses such a specification to generate step by step the next action to perform and, optionally, the set of skills to be active, considering the current state of the environment.
- The *action specialist* verifies the physical feasibility of actions and predicts the performance of actions (e.g., time required, distance to cover, required charge of battery, etc.).
- The *trajectory planner* generates collision-free trajectories that can be followed by the drone.
- The *yaw planner* generates a yaw value for the aerial platform specified either as a point to look or as a yaw to look.

The planning system communicates with processes that perform the following tasks:

- The *manager of actions and skills* accepts the actions and skills requested by the mission planner, expressed as symbolic descriptions (e.g., take off, move to a point, etc.), and translates them into specific orders for the motion controllers and the activation of certain running processes. It responds asynchronously to the requests.
- The *action monitor* supervises the correct execution of initiated actions and detects either when they are completed or when they have failed. It works synchronously, monitoring the actions actively.
- The *event detector* detects the presence of significant events to be considered by the mission planner (e.g., the recognition of a particular visual marker). It synchronously monitors the considered events.
- The *problem manager* collects errors and initiates the corresponding recovery actions. The detected problems are informed to the mission planner to react in consequence. It watches out for new problems synchronously.

We reused the trajectory planner and the yaw planner (see [10] for details about these components) and we adapted the

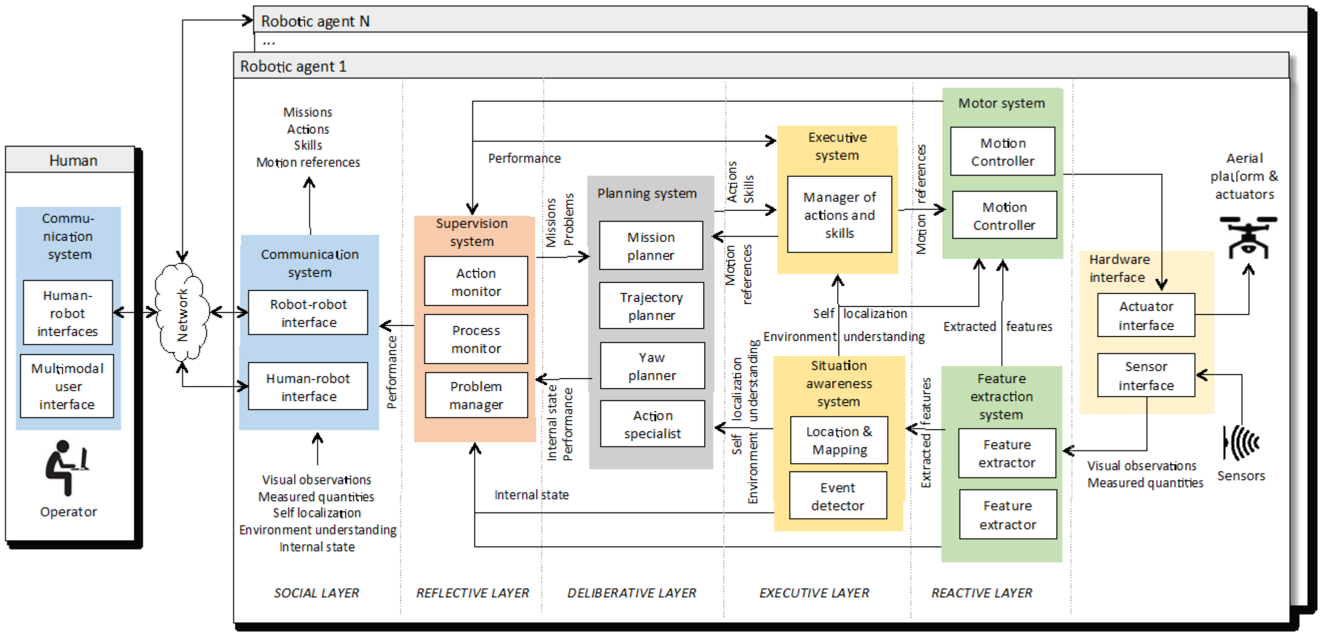

Figure 1: Main components of the Aerostack multi-layered architecture [4]

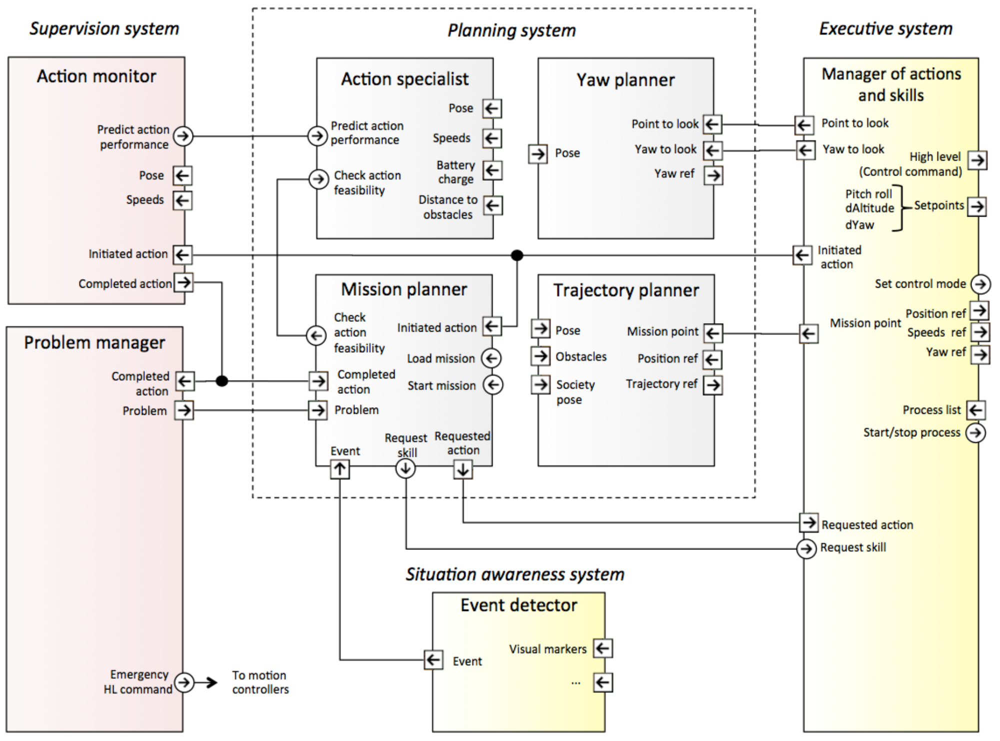

Figure 2: Detail of processes used in Aerostack to interpret the TML mission specification language.

manager of actions and skills. Since Aerostack is a software framework platform independent, the resulting mission planning system also has this property. Only certain platform-dependent parameters must be configured.

5 Evaluation

This section presents the results of evaluations that we performed for the TML language: (1) flight tests and (2) mission specification for the IMAV 2016 competition.

5.1 Flight tests

The TML language was tested in real flight using different mission cases with various degrees of complexity. This section describes one of these cases, whose formulation was based on a surveillance mission. The goal of this mission is to search for a subject in a spatial area avoiding any type of obstacle. When the aerial robot recognizes the presence of the subject, it performs an alert action (e.g., notify the location of the object, start an alarm sound, etc.) and returns to the starting point.

For the sake of simplicity, we defined an indoor mission where the robot performs a search routine (a squared trajectory with a side of 2 meters) and detects obstacles or the searched subject using visual markers (ArUco markers). These simplifications facilitate the practical development of the real flight and allow testing the functionality of the TML language. The experiment was carried out with the aerial platform Parrot AR Drone 2.0. We used the same specified mission considering several scenarios with changes in the environment with different locations of obstacles and searched subject. Figure 3 shows graphics representing the trajectories followed by the robot in the different scenarios.

The described experiment demonstrates that the TML language can be used to specify and execute a mission for an aerial vehicle in a dynamic environment. The experiment shows how the robot adapts correctly its behavior to changes in the environment, considering different locations of obstacles and the searched subject.

It is worth to clarify that this behavior was achieved in two different decision planes. On the one hand, the process *mission planner* supports an adaptive behavior corresponding to the moment when the subject is detected, according to the specified reaction written explicitly in TML language (return to the starting point).

On the other hand, the process *trajectory planner* supports an adaptive behavior corresponding to the presence of obstacles, generating collision-free trajectories to be followed by the robot. In summary, this experiment corresponds to a representative type of mission based on surveillance but presents also similarities to other types of missions (e.g., search and rescue, terrain mapping, etc.) where the aerial robots must show certain degree of autonomy.

5.2 IMAV 2016 Competition

We analyzed the ability of the TML to be used as specification language for the mission of the IMAV 2016 indoor competition. This mission includes partial goals such as taking off from a moving platform, entering the building via different entrance (doorway, chimney or window), pickup objects and release them to the correct locations, exiting the building, landing on the moving platform, and mapping of the indoor environments.

Appendix A shows an example of specification that we wrote in TML language for this mission. For this specification, we made certain assumptions about actions and skills. For example, we assumed that the robot is able to do specific actions (pick up item, drop item), specific skills (recognize bucket, recognize item), or general actions (go to a rectangle identified by a colored frame, like the door in the competition, or explore the space building a map).

According to these assumptions, we were able to write in TML the corresponding IMAV 2016 mission, which demonstrates that this language provides enough expressive power to formulate such a type of missions. In particular, tasks were useful to structure modularly the different parts of the mission: take off at operating zone, enter building, explore building, etc. Event handlers were useful to react to recognized items and buckets while the robot is exploring the building. The concept of actions and skills were also useful to model the different abilities of the robot for this mission.

6 Conclusion

In this paper, we have described TML, a language that follows a task-based approach to specify missions for aerial robotics. The TML language uses the concept of task as a basic notion to structure the mission and other intuitive concepts (actions, skills, events) that can help to be easily learned and used by operators.

We implemented a TML interpreter as part of the Aerostack software architecture. Thanks to the use of this framework, the implementation of the TML interpreter is independent on aerial platforms and able to be used in real flights. The TML interpreter is freely available for the research community as part of the Aerostack architecture (www.aerostack.org). The evaluation of TML showed that this language is expressive enough to specify a mission such as the one defined for the IMAV 2016 competition. The real flight experiments with the interpreter of TML in Aerostack showed also an adequate adaptation to environment changes.

Our future work includes adding improvements to this language such as new and more complex actions and skills. We also plan to build a graphical user interface that help operators to specify missions based on TML. This interface can use graphical notations and restricted menu options. Another line of future work is to combine TML with a multi-agent approach to support distributed and coordinated planning (e.g., [11]).

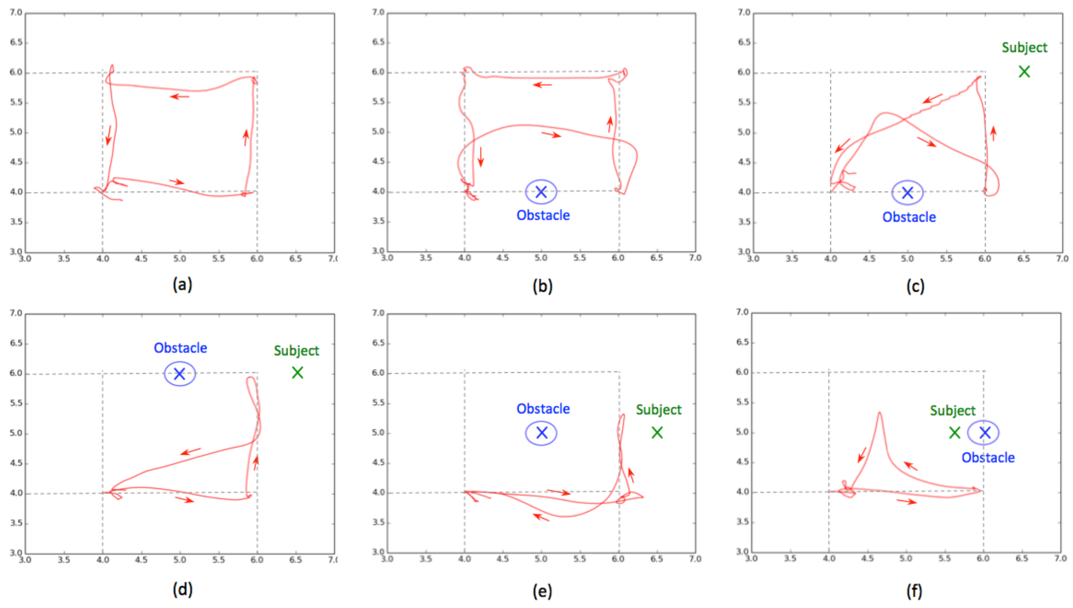

Figure 3: This figure shows different trajectories followed by our aerial robot in real flights during the execution of the same mission specified in TML language. The experiment shows how the robot adapts correctly its behavior to changes in the environment, considering different location scenarios of an obstacle and the searched subject. In the figure, the origin of the movement is the point (4.0, 4.0). The graphic (a) corresponds to the squared trajectory followed by the robot in a scenario without any influence from the environment.

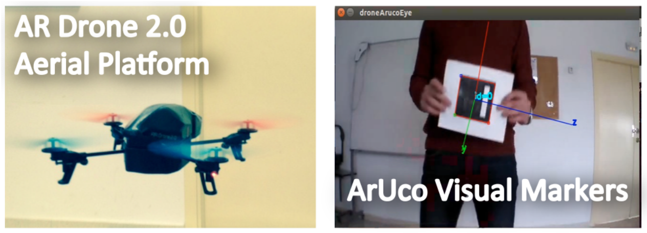

Figure 4: Platform and markers used in our experiments.

Acknowledgements

This research work has been supported by the Spanish Ministry of Economy and Competitiveness through the project VA4UAV (Visual autonomy for UAV in Dynamic Environments), reference DPI2014-60139-R.

References

[1] E. Santamaria, P. Royo, C. Barrado, E. Pastor, J. Lpez, and X. Prats. Mission aware flight planning for unmanned aerial systems. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Honolulu (HI), 2008.

[2] B. Schwartz, L. Nagele, A. Angerer, and B. A. MacDonald. Towards a graphical language for quadrotor missions. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2014.

[3] A. L. Rothenstein. *A Mission plan specification language for behaviour-based robots*. PhD thesis, University of Toronto, 2002.

[4] J. L. Sanchez-Lopez, R. A. Suarez Fernandez, H. Bavle, C. Sampedro, M. Molina, J. Pestana, and P. Campoy. Aerostack: An architecture and open-source software framework for aerial robotics. In *The 2016 International Conference on Unmanned Aircraft Systems ICUAS*, Arlington, VA, 2016.

[5] R. C. Arkin, E. M. Riseman, and A. Hansen. Aura: An architecture for vision- based robot navigation. In *proceedings of the DARPA Image Understanding Workshop*, Los Angeles, CA, 1987.

[6] E. Gat. On three-layer architectures. In David Kortenkamp, R. Peter Bonnasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots*. AAAI Press, 1998.

[7] A. Sloman. What sort of architecture is required for a human-like agent. In M. Wooldridge and A. Rao, editors, *Foundations of Rational Agency*. Kluwer Academic Publishers, 1999.

[8] R. J. Brachman. System that know what they're doing. *IEEE Intelligent Systems*, 17(6):67–71, 2002.

[9] B.R. Duffy, M-Dragone, and D.M.P. O'Hare. The social robot architecture: A framework for explicit social interaction. In *In Proceedings Cognitive Science Workshop, Android Science-Towards Social Mechanisms*, Stresa, Italy, 2005.

[10] J.L. Sanchez-Lopez, J. Pestana, P. de la Puente, R. Suarez-Fernandez, and P. Campoy. A system for the design and development of vision-based multi-robot quadrotor swarms. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 640–648, May 2014.

[11] C. Sampedro, H. Bavle, J. L. Sanchez-Lopez, R. A. Suaarez-Fernandez, A. Rodrguez-Ramos, M. Molina, and P. Campoy. A flexible and dynamic mission planning architecture for uav swarm coordination. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, Arlington, VA, 2016.

APPENDIX A: MISSION FOR IMAV 2016

This appendix shows a mission specification in TML language for IMAV 2016 Indoors Competition. This corresponds to an example where only one item is picked up.

```
<mission name="Indoor Competition IMAV 2016">
  <skill name="AVOID_OBSTACLES"/>
  <!-- TASK: Take off at operating zone -->
  <task name = "Take off at operating zone">
    <skill name="SELF_LOCATE_BY_MARKERS"/>
    <task name = "Take off to start the mission">
      <action name="TAKE_OFF"/>
    </task>
    <task name = "Memorize operating zone">
      <action name="MEMORIZE_POINT"/>
         <argument name="coordinates"
                   label="operating zone"/>
    </task>
  </task>
  <!-- TASK: Enter building -->
  <task name = "Enter building">
    <task name = "Go to the door">
      <action name="GO_TO_RECTANGLE">
        <argument name="frame color" value="red"/>
      </action>
    </task>
    <task name = "Enter through the door">
      <action name="MOVE_FORWARDS">
        <argument name="distance" value="1.5"/>
      </action>
    </task>
    <task name = "Memorize where is the door">
        <action name="MEMORIZE_POINT">
            <argument name="coordinates"
                      label="door coordinates"/>
        </action>
    </task>
  </task>
  <!-- TASK: Explore building -->
  <task name = "Explore building">
    <skill name="RECOGNIZE_ITEMS"/>
    <skill name="RECOGNIZE_BUCKETS"/>
     <action name="EXPLORE_SPACE">
       <argument name="build map" label="yes"/>
    </action>
  </task>
  <!-- TASK: Transport item A -->
  <task name = "Transport item A">
    <condition label="item A coordinates"
               comparison="known" value="yes"/>
    <condition label="bucket A coordinates"
               comparison="known" value="yes"/>
    <task name = "Go to item A">
        <action name="GO_TO_POINT">
            <argument name="coordinates"
                      label="item A coordinates"/>
        </action>
    </task>
    <task name = "Pick up item A">
        <action name="PICK_UP_ITEM"/>
    </task>
    <task name = "Go to bucket A">
        <action name="GO_TO_POINT">
            <argument name="coordinates"
                      label="bucket A coordinates"/>
        </action>
    </task>
    <task name = "Drom item A">
        <action name="DROP_ITEM"/>
    </task>
  </task>
  <!-- TASK: Exit building -->
  <task name = "Exit building">
    <task name = "Return to the door">
      <action name="GO_TO_POINT"/>
         <argument name="coordinates"
                   label="door coordinates"/>
    </task>
    <task name = "Correct orientation">
      <action name="ROTATE_YAW"/>
        <argument name="angle" label="90"/>
    </task>
    <task name = "Exit building through the door">
      <action name="MOVE_FORWARDS"/>
        <argument name="distance" value="1.5"/>
    </task>
  </task>
  <!-- TASK: Land at operating zone -->
  <task name = "Land at operating zone">
    <skill name="SELF_LOCATE_BY_MARKERS"/>
    <task name = "Go to operating zone">
      <action name="GO_TO_POINT"/>
         <argument name="coordinates"
                   label="operating zone"/>
    </task>
    <task name = "Final landing">
      <action name="LAND"/>
    </task>
  </task>
  <!-- Event handlers -->
  <event_handling>
  <!-- EVENT: Item A found -->
  <event name="Item A found">
      <condition current_task="Explore building"/>
      <condition parameter="Item A found"
                 comparison="equal" value="yes"/>
      <action name="MEMORIZE_POINT">
        <argument name="coordinates"
                  label="item A coordinates"/>
      </action>
      <termination mode="CONTINUE"/>
    </event>
  <!-- EVENT: Bucket A found -->
    <event name="Bucket A found">
        <condition current_task="Explore building"/>
        <condition parameter="Bucket A found"
                   comparison="equal" value="yes"/>
        <action name="MEMORIZE_POINT">
            <argument name="coordinates"
                      label="bucket A coordinates"/>
        </action>
        <termination mode="CONTINUE"/>
    </event>
  </event_handling>
</mission>
```

Example 4: Mission specification file for IMAV 2016.