ON BOARD AUTONOMOUS OPERATIONS FOR OPS-SAT

Simone Fratini, Julian Gorfer[*], Nicola Policella, and Alessandro Donati

Artificial Intelligence and Operation Innovation Group
European Space Agency, ESA-ESOC
Robert-Bosch-Strasse 5, 64293, Darmstadt, Germany
name.lastname@esa.int

ABSTRACT

Upcoming missions, either targeting Earth, planets or deep space, will require a higher degree of on-board autonomy operations to increase quality science return, to minimize close-loop space-ground decision making and to enable new scenarios. This paper describes a new planning and execution architecture deployed to validate on-board autonomous capability for the ESA OPS-SAT mission. The architecture integrates a domain-independent decision making system, an executor and an image classification procedure, to enable a full, on-board, sense-plan-act loop.

Key words: On-Board Autonomy, Planning & Scheduling, Data Analytics.

1. INTRODUCTION

At a very high level, the mission of scientific satellites in space consists in observing and in downloading collected data to ground stations. Current common practice for most of these missions, is to generate plans on the ground in the mission control facilities, and then upload them as a static sequences of time-tagged telecommands that must be executed by the satellite at some point in the future. This approach has obvious limitations because of the intrinsic uncertainty at execution time: for instance in case of Earth observing satellites, meteorological conditions can significantly impact observations quality and the amount of data actually generated by the experiments are rarely known in advance with enough precision to generate accurate dump plans.

As a result, the efficiency of these systems is reduced by contingent conditions difficult to predict at planning time and difficult to accommodate in a rigid process that keeps planning and execution strictly distinct: a lot of low quality data is being collected and dumped to ground station for instance, the amount of planned experiment is reduced because of the use of very conservative models for memory management and so on. Besides that, opportunistic science can't be achieved because of the communication delays (especially for deep space missions), with the result that spontaneous phenomena can't be observed properly, mainly because when they are detected on ground analyzing dumped data it is usually too late to organize a proper observation plan.

Upcoming missions instead, either targeting Earth, planets or deep space, are requiring a higher degree of on-board autonomy operations to increase quality science return, to minimize close-loop space-ground decision making, to enable new scenarios. Artificial Intelligence technologies like Machine Learning and Automated Planning are becoming more and more popular as they can support data analytics conducted directly on-board as input for the on-board decision making system that generates plans or updates them while being executed. This paper describes a new on-board planning and execution architecture, completely deployed at ESA, to target this need of autonomy for the OPS-SAT mission, a 3-Unit CubeSat structure. The objective is to test and validate the introduction of new techniques in mission control as well as in on-board systems [22]. OPS-SAT consists of a satellite that is only 30cm high but equipped with an experimental computer that is ten times more powerful than any current ESA flying spacecraft.[1]

The architecture presented is based on three pillars: a domain independent planner, a platform-agnostic executive, and an on-board image classification system. This takes inspiration from previous NASA-JPL missions that have pioneered and proven the value of AI planning and scheduling for injecting autonomy in space applications: Remote Agent Experiment (RAX) on Deep Space 1 (DS-1) [19] and the Autonomous Sciencecraft Experiment (ASE) on Earth Observing 1 (EO-1) [8]. Lately, NASA launched the IPEX CubeSat [7] to validate new technologies for on-board image processing and autonomous operations. ESA contributed in this area by promoting activities for supporting on-board autonomy for robotics – see GOAC, Goal Oriented Autonomous Controller [5] and subsequent evolution, GOTCHA[21] (GOAC TRL In-

[*] Work performed at ESA under traineeship activity.

[1] OPS-SAT launch date is currently scheduled for October 2019.

crease convenience enhancements hardening and application extension), that gave inspiration for this proposal.

The paper first introduces the mission scenario and the autonomy experiment. Then it presents the architecture implemented and describes one of the testing scenarios used to validate on-board autonomous capability on OPS-SAT.

2. OPS-SAT

The OPS-SAT mission has been conceived as an opportunity for testing new concepts and technology in an in-flight environment. The idea is that the validation through OPS-SAT would make easier the acceptance and adoption of these solutions in future missions. This is mainly because of the difficulty to perform live testing in the domain of mission control systems. No one wants to take any risk with an existing, valuable satellite unless strictly necessary. This makes very complex to introduce new concepts, techniques or systems in orbit. OPS-SAT is based on a low-cost satellite that is rock-solid safe and robust even if there are any malfunctions due to testing. The robustness of the basic satellite itself will give ESA flight control teams the confidence they need to upload and try out new, innovative control software submitted by experimenters; the satellite can always be recovered if something goes wrong during an experiment execution. Achieving this level of performance and safety at a low cost is a challenge. To do this, OPS-SAT combines off-the-shelf subsystems as typically used with cubesats, the latest terrestrial microelectronics for the on-board computer, and the experience ESA-ESOC has gained in safely operating satellites for the last 40 years. OPS-SAT can be seen as an open, flying laboratory for in-orbit demonstration of (r)evolutionary new control concepts and software systems that would be otherwise too risky to trial on a "real" satellite.

2.1. On-board Autonomy Experiment

Our on-board autonomy experiment is based on two pillars: (1) the possibility of analyzing images directly on-board and (2) the presence of an intelligent planner to promptly react to the results of the analysis by accommodating the current plans and/or to generate new plans for achieving new goals. In particular, the goal of our experiment is not limited to validate the adoption of on-board autonomy capabilities in the case of planning satellite tasks, but it also considers to evaluate the necessary changes to be introduced in the ground system to properly control the satellite without limiting the on-board intelligence.

A superficial analysis could assume that functions usually performed on ground will disappear from the ground segment once performed on-board. However a closer look shows that autonomy induces three evolutions in the role of the Ground Segment [13]:

1. Commanding and monitoring of on-board autonomous (decisional) processes. This can be quite different from Monitor & Control (M&C) of classical processes (i.e. how to M&C on-board orbit control, on-board mission planning or advanced Fault Detection, Isolation and Recovery (FDIR), etc.);

2. On ground automation, that participates to the global system autonomy (i.e. automation of Telemetry/Telecommands loop, mission plan update, detailed telemetry downloading);

3. Support to on-board autonomous processes. Enhanced autonomy induces a new re-partition of processes between Ground and Space, but even if some decisions are taken on-board, it is still valuable that part of it is performed on ground (i.e. for FDIR, platform management, orbit management).

The latter is crucial for mitigating the risks associated with on-board autonomy. All possible situations cannot be tested before launch, and on-board constraints usually do not allow embarking the most capable software. In that case, the ground segment shall provide functions to handle extreme situations.

Testing scenario. The potentialities of the architecture for on-board autonomy presented in this paper are demonstrated in a testing scenario which is composed by two experiments of increasing complexity.

In the first experiment OPS-SAT is provided with a target latitude and longitude coordinates to observe in a given time interval (identifying multiple opportunities of target visibility). The goal of the experiment is to schedule the observations, analyze the images to asses if the target is visible,[2] store, and download the images. When a picture taken results not usable, it is discarded and a new opportunity should be scheduled when the target will be visible again (in following orbits). If the picture cannot be successfully taken in the specified time interval, the task is discarded. The objective of the experiment is to demonstrate autonomous capabilities to schedule and observation and to download only meaningful observations.

The second experiment is divided in two phases. In the first phase the satellite is continuously scouting for an interesting phenomena (e.g., the presence of a volcano) by using an on-board image classification algorithms. In practice the satellite will repeatedly take pictures and analyze them until a particular *target* is recognized in the picture. During this scouting phase, the pictures will not be stored in the memory nor downloaded to the ground. Only the coordinates and the type of target identified is returned.

[2]Depending on weather conditions, an image can be "too cloudy" and the target invisible.

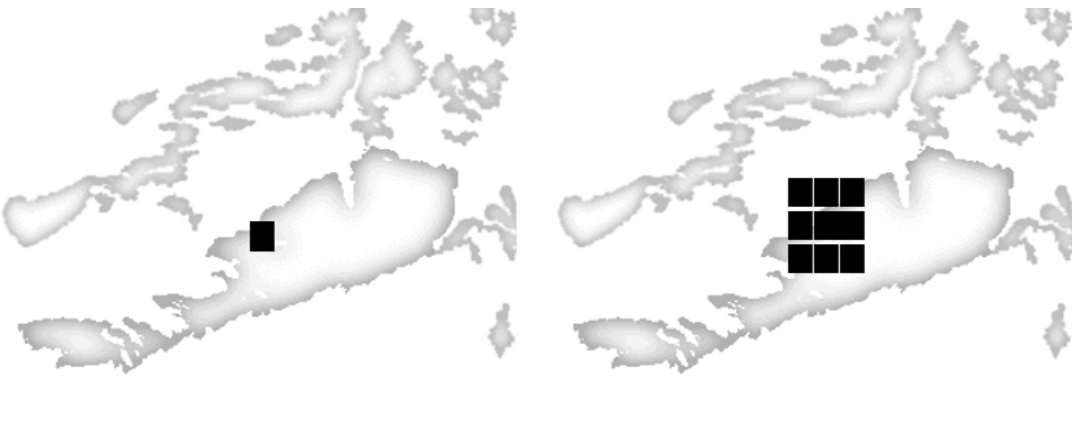
(a) Target Identification (b) Area of Interest

Figure 1. Survey of the Area of Interest

The finding of the target (see example in Fig. 1(a)) stops the first phase and triggers a second one in which a survey of the *Area of Interest* (AoI) is planned. This consists in planning a series of pictures in order to cover a larger area around the recognized target, Fig. 1(b). Each picture of the survey will have latitude and longitude values such that the area around the identified target is consistently covered. More precisely, given a target identified by the couple of latitude and longitude values (x, y), the survey of the area consists of a set of actions $TakePicture(x', y')$ to take pictures at latitudes (x', y') around (x, y)[3]. The objective of the second experiment is to demonstrate opportunistic science capabilities and not trivial on-board planning and control capabilities.

In the allocation of the pictures several aspects shall be considered. First, the satellite orbit: in fact, given the geographic location of each picture, the orbit is used to determine the time windows when the satellite has direct line-of-sight to the picture's location. Second, the limited on-board memory: each picture has to be stored in the on-board memory before it can be downloaded to the ground (no continuous communication to earth is possible). Last, the limited availability of ground stations to download the data. This can be seen as a set of downloading activities which routinely empty the on-board memory (completely or partially depending of their duration). This resulting planning problem requires to find a time allocation of a set of pictures, $TP = \{TakePicture(x^0, y^0), .., TakePicture(x^n, y^n)\}$, considering that:

- each picture has a set of windows of opportunity, O_i;
- the capacity of the on-board memory, C_j;
- a set of pre-defined downloading opportunities $D = \{dl_0, .., dl_m\}$.

It is important to notice that the problem does not imply any temporal order in which to take the different pictures composing the survey. Furthermore, during the execution of the plan, further adjustment could be required. In particular we will consider to analyze directly on-board the quality of the single pictures taken from the AoI. For instance cloudy pictures could be of no use, therefore the plan should be updated to re-take the picture.

The experiment follows the path first investigated in [8]: where in [8] a scheduler is used on-board to allocate set of flexible activities, in our case a more complex automated planner is used to also generate on-board the activities to achieve high level survey goals.

3. SYSTEM ARCHITECTURE

Figure 2 illustrates the system architecture. This is composed by (1) an automated planner that provides the deliberative capabilities (in green); (2) an executive module to instantiate plans into commands for the platform, providing controlling and execution monitoring capabilities (in pink); (3) a set of data analyzer and anomaly detection modules (in blue); (4) an overall controller that masters the interactions among the planner, the executive and the data analyzers.

The Command and Telemetry Dispatcher or CTD (in black in Fig. 2) is responsible for connecting the system to the platform, by interfacing to a middleware layer, the Nanosat Mission Operation Framework, or NMF [9]. The framework, highlighted in yellow in the figure, wraps the different OPS-SAT's hardware components and modules such as the GPS receiver, the HD-Camera Module, or the Magnetometer. The autonomy system can directly manipulate a subset of these components and read sensory values with simple Java API method calls. Changing the

[3]Considering the distance between two adjacent images, δ_{lat} and δ_{long}, and $level$ the number of 'circles' around the target that should be analyzed, we have, $\forall i, j = -level, ..., level$:

$$x' = x + i * \delta_{lat}$$
$$y' = y + j * \delta_{long}$$

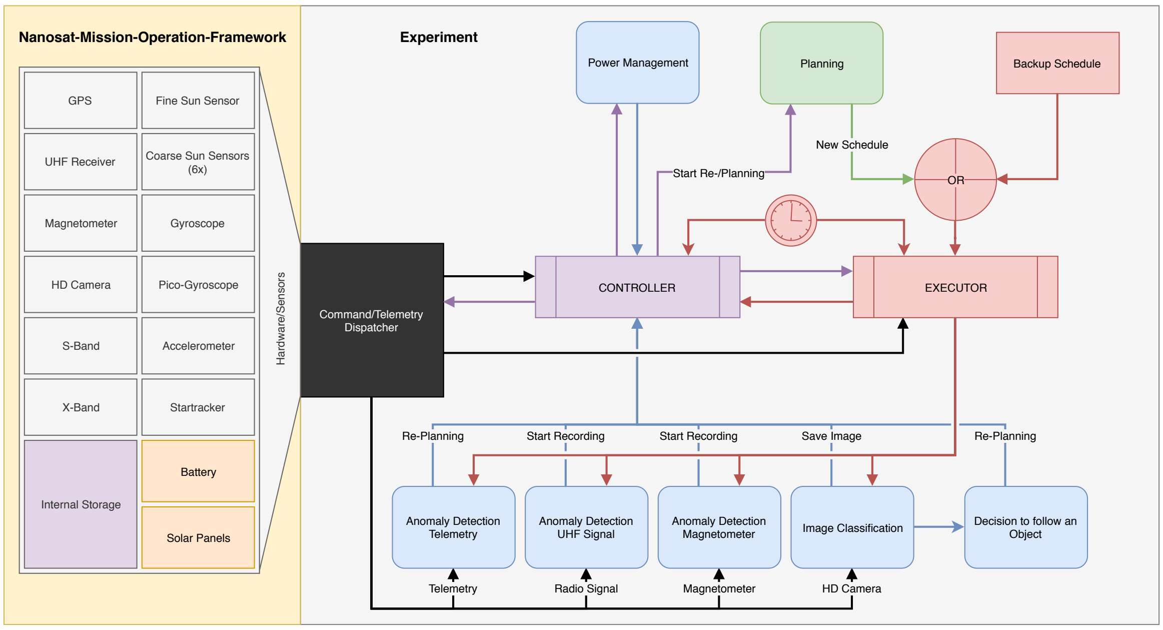

Figure 2. System Overview.

spacecraft attitude or taking pictures are examples of the functionality provided by the NMF.

The CTD translates control statements received from the Executor into NMF procedure calls, and returns parameter values to telemetry queries from the Executor and/or the Controller. For instance when a picture is taken, the Controller uses the CTD to send it to the Image Classification module. The latter performs a different tasks depending on the phase of the experiment. During scouting, the image classifier check the presence of a particular event (e.g., a volcano). During survey instead, the image classifier focuses only on the quality of the picture taken. In both phases the results from the image classification analysis are sent to the controller that can take different decisions: (1) save the image and plan a survey if a target is found; (2) do nothing if the target is not in the image; (3) do not save the image if cloudy and plan a new acquisition for the picture.

3.1. Deliberative Technology

The deliberative capabilities are provided by a planning technology based on constraint based temporal planning with timelines. The planner has been developed on top of APSI (Advanced Planning and Scheduling Architecture) [3], an ESA software framework designed to improve the cost-effectiveness and flexibility of AI planning and scheduling tools deployment.

Timeline-based temporal planning is inspired by classical Control Theory, where a problem is modeled as a set of entities, or *timelines*, whose properties vary in time, such as one or more physical subsystems. The timelines evolve over time concurrently while their behaviors can be affected by control decisions[4].

Specifically the planning problems is defined using two classes of modeling components, *state variables* and *resources*, and their valid interactions are specified by means of *synchronizations*.

State variables represent components that can take sequences of symbolic states subject to various (possibly temporal) transition constraints. This primitive allows the definition of *timed automata* representing the constraints that specify the logical and temporal allowed transitions of a timeline. A timeline for a state variable is valid if and only if it represents a *timed word* accepted by the automaton. The timed automaton (or in the APSI case, the state variable) is a very powerful modeling primitive, widely studied [2], and for which different algorithms exist to find valid timelines.

Resources are used to model any physical or virtual entity of limited availability, like those commonly used in constraint-based schedulers [4]. A resource timeline represents the resource profile of availability over time as resulting from the quantitative use/production/consumption over a time interval or at time instants. A resource timeline is valid if and only if the profile never exceeds the resource capacity.

The physical and technical constraints that influence the interaction of the sub-systems (modeled either as state variables or resources) are represented by temporal and logical synchronizations among the values taken by the automata and/or resource allocations on the timelines.

[4]A detailed description of the timeline based approach, state of the art of the technologies in use and basic concepts is out of the scope of this paper. More information can be found, for example, in [17, 11, 12, 6].

Conceptually these constructs define valid schema of values allowed on timelines and link the values of the timelines with resource allocations. In particular they allow the definition of Allen's relations [1] like quantitative temporal relations among time points and time intervals as well as constraints on the parameters of the related values. From a planning perspective, the synchronizations define the cause-effect relationships among the states of the system modeled, describing how a given status can be achieved.

In this context, problem solving consists of generating feasible timelines to model a desired system behavior, simulating the temporal evolution of relevant subsystems status and of relevant parameters values. Timelines are then used to control the platform in a closed-loop control schema: an *executor* translates timeline's values into commands and synchronizes actual values read from the platform with the simulated values into the timelines. When discrepancies are detected, a new plan is generated starting from the actual status of the platform.

One of the reasons for the use of this type of planning is the capability to enable, in a *flexible* way, the integration of planning and scheduling tasks. A plan is constituted by a set of timelines and temporal/value constraints among them. A timeline is a sequence of values, a set of ordered transition points between the values and a set of distance constraints between transition points. When the transition points are bounded by the planning process (lower and upper bounds are given for them) instead of being exactly specified we refer to the timeline as *time flexible* and to the plan resulting from a set of flexible timeline as a *flexible plan*.

As many timeline-based planning systems, the APSI framework use Simple Temporal Networks (STN)[10] to test the consistency of partial plans generated during the search process and to represent the temporal information for solution plans. These systems produce flexible plans where every solution to the final STN provides an acceptable schedule. Previous control systems based on APSI as deliberative layer (like [5] or [20]) guarantees *dispatchability* of the plan as defined in [18] by propagating full or decoupled temporal networks during execution. In situations where everything is under the control of the agent driven by the plan, or a careful modeling can reduce the impact of uncertainty, a dispatchable plan is enough to ensure a reliable control.

Many application domains, however, involve temporal uncertainty on the duration of some processes or on the occurrence of some events that can't be removed simply with careful modeling[5]. In these cases, the values for the variables that are under the agents control may need to be chosen at execution time so that they do not constrain uncontrollable events whose outcomes are still in the future: the *controllability* problem. In these cases, the plan's temporal network needs to be post-processed, to tighten planned bounds of controllable events and to calculate a

[5]Or that would result in cumbersome or extremely expensive modeling.

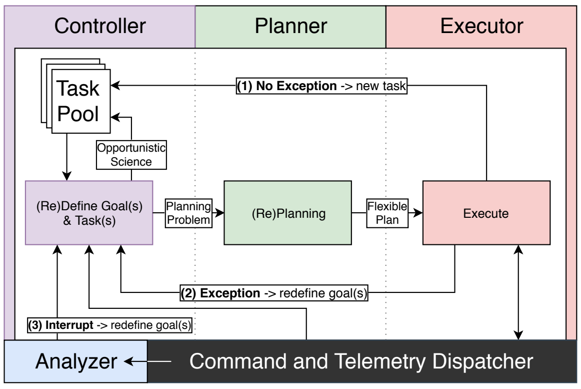
Controller
Planner
Executor
Task Pool
(1) No Exception -> new task
Opportunistic Science
(Re)Define Goal(s) & Task(s)
Planning Problem
(Re)Planning
Flexible Plan
Execute
(2) Exception -> redefine goal(s)
(3) Interrupt -> redefine goal(s)
Analyzer
Command and Telemetry Dispatcher

Figure 3. Control Schema

strategy to provide the executor of enough information to generate a valid/consistent schedule to any situation that may arise in the external world (*strong* controllability) or, if this is not possible, to choose future controllable events given the contingent occurrences observed in the past, the *dynamic* controllability. Dynamic controllability can be checked in $\mathcal{O}(\mathcal{N}^3)$, generating dispatchable STNs from projections of dynamic controllable plans[16].

In the OPS-SAT experiment some events presents uncertainty in their temporal duration (see Section 4 for some examples), hence the Planner generates temporally flexible plans and leave to the Executor the task of instantiating the schedule at run time (see Section 3.3).

3.2. Controller

The Controller is responsible for integrating the planner with the executive and the analyzer. Figure 3 shows the control workflow. The controller has a pool of tasks. In our current testing scenario three main tasks are considered: (1)$TakePicture(x, y, t_s, t_e)$ to take pictures at latitudes (x, y) in the time interval $[t_s, t_e]$; (2) $Survey(x, y, l)$ to survey an Area of Interest of l circles around (x, y); (3) $Scout(t_s, t_e)$ to scout an area in the time interval $[t_s, t_e]$.

Once the task is selected, the controller monitors the current status of the platform, via the CTD, and defines a planning problem with goals and initial conditions for the planner. The Planner creates a flexible plan that is then passed to the Executor.

The Executor works with the CTD to dispatch commands and to inject values into the plan. Three possible situations can occur: (1) the execution goes as planned, i.e. the flexibility provided by the plan is sufficient to achieve the goals. In this case the controller is notified with success and the cycle restart from the pool of tasks; (2) during execution an exception triggers a re-planning, i.e. some temporal occurrences goes out of the planned bounds or values not planned are injected into the timelines. In this case the execution aborts and the control is given back to the Controller, that is in charge of generating a new sets of goals. In the testing scenario a no-nominal situation occurs after taking an image when it is classified as cloudy, hence a replanning occurs after discarding the image; (3) an interrupt from one of the analyzers triggers a re-planning. In this case the plan is being executed as nominal, but an external trigger fosters the autonomous generation of new tasks. In this case the Controller can either interrupt the execution and generate a new plan or decide to generate only the tasks and wait for them to be selected in following iterations.

3.3. Executive

The executive is organized into three core services: the Clock, the Plan Executor and the Timeline Executor. The Clock (shown as a clock icon in Figure 2) generates ticks in a periodic interval and sends notifications to synchronize the system real time with the plan virtual time in the threads under concurrent execution. A tick contains an increasing id number and the timestamp of its release, triggering the Controller and the Executor. In this way it can be determined how much time passed since the last trigger and if a tick was accidentally skipped because a thread were unable to process it.

The Executor process is initiated by the Controller that passes a flexible plan in which each timeline provides a set of transitions between values to be scheduled with respect to the transitions of the other timelines. While controllable transitions are under the control of the Executor, the uncontrollable ones are translated into queries for the CTD that will inform the Executor on their occurrence. For each timeline, the Executor creates a separate thread and monitors the execution ensuring controllability, propagating events as constraints on the temporal database or on the values in the timelines. Once all the threads have reached the end, the Executor sends a notification to the Controller that the plan has been successfully completed. Conversely, if an exception arises, the Executor raises an exception for the Controller and aborts the process.

3.4. Data Analyzers

The analyzers implemented are based on Convolutional Neural Network for image classification (see for instance [14] for a survey). In particular several neural networks have been created to analyze information coming from the NMF platform.

A first model has been introduced to analyze images and determine the level of cloudiness. A small Convolutional Neural Network (see Fig. 4), which has two Convolutional Layers followed by Maxpooling Layers and two fully connected Dense Layers, can classify images to be cloudy/clear with an accuracy of 0.937 and an F1-Score of 0.935 (see Table 1). However, the model misclassifies an image as cloudy in case of presence of ice or snow. Table 2 shows the confusion matrix of the validation set.

Accuracy	0.937
Precision	0.939
Recall	0.932
F1-Score	0.935

Table 1. Precision, Recall and F1-Score from the Validation Set.

		Prediction	
		Clouds	**Clear**
True	**Clouds**	80	3
Label	**Clear**	6	54

Table 2. Confusion Matrix of the Validation Set.

The six misclassified cloud images are all images that contain a high percentage of white pixels due to snow and ice. A possible explanation could be that the image gets downscaled from 2000x2000 pixels to 128x128 pixels which leads to an information loss. Also OPS-SAT camera can only take RGB images and has no other channels available to distinguish between white clouds and white snow. The three misclassified pictures with no clouds on the other hand could be misclassified because of inconsistent labelling. To solve the problem of misclassifying snow and ice images as cloudy, the model needs more data. It would also help to have other channels available to distinguish clouds from snow and ice.

The second model classifies an image to identify interesting objects to track. The same small Convolutional Neural Network can be used to classify points of interest in images. The only difference to the previous model is the bigger output layer. Each node of the output layer refers to a class. A preliminary concept of this model classifies following labels: (1) Volcano, (2) Island, (3) Human-made, (4) Land, and (5) Water. One image can have multiple classification labels. For example an image containing an island also contains water.

4. MODEL

For the case of the Testing Scenario described in Sect. 2.1, we need to control the camera payload, the memory storage sub-system and the attitude of the satellite. Relevant information out of control of the planning system but needed at planning time is the GPS coordinate covered by the satellite in a given time interval. Besides that we model also the image classification sub-system.

The tasks to be executed (of the three types defined in in Sect. 3.2) are selected by the Controller from the Task Pool in Figure 3. Tasks are translated by the Controller in APSI planning problems, made of a set of planning statements

$$\text{TAKEPICTURE}(?x, ?y, ?file\_id, ?store, ?identify)$$

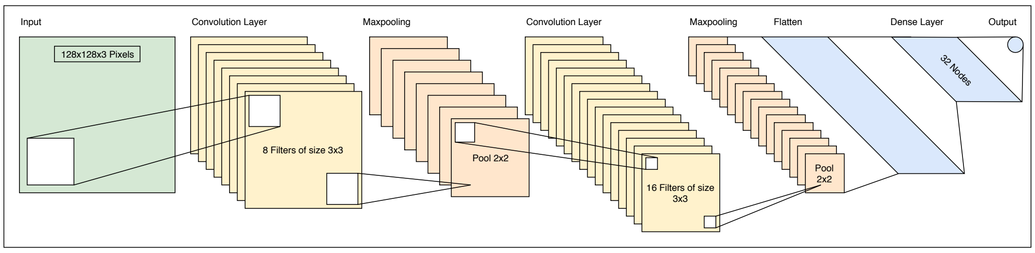

Figure 4. Convolutional Network

to be decomposed by the planner into lower level activities to take a picture in $(?x, ?y)$. Once executed, the platform provides a value for $file\_id$, the ID assigned to the picture. $?store$ and $?identify$ are two Boolean variables to set whether the picture has to be stored and/or a target identification process has to be run on the picture. In addition to that, given the planning statements in which the task has been decomposed, the planning problems contains the computed visibility windows for the coordinates to be observed and the initial value of allocated on-board memory.

The task $TakePicture(x, y, t_s, t_e)$ for instance, to take pictures at latitudes (x, y) in the time interval $[t_s, t_e]$, it is decomposed into a single $takePicture(x, y, ?file\_id, T, F)$ statement, to take a picture in (x, y) and store it $(?store = T)$. No need to identify targets for this task, then $?identify = F$. The visibility windows for the target position (x, y) are identified in the time interval $[t_s, t_e]$.

A task $Scout(t_s, t_e)$ to scout an area in the time interval $[t_s, t_e]$ it is decomposed into a set of n statements $takePicture(x_i, y_i, ?file\_id, F, T)$, where the latitudes (x_i, y_i) are computed by subdividing the time interval $[t_s, t_e]$ into n intervals and calculating which latitudes are covered by the satellite in that interval. In this case we do not store the picture taken (hence $?store = F$), but we want to identify a target (then $?identify = T$).

The planning model is a standard timeline-based planning model, with a *Mission Timeline* MT where we post the goals describing the objectives to be achieved with the plan. The camera is modeled as a state variable CAM, the attitude is modeled by means of a state variable ATT taking values LOCKED$(?x, ?y)$, when the satellite is actively pointing the target in $\langle ?x, ?y \rangle$ and following it along the orbit, UNLOCKED$(?x, ?y)$, when the satellite is pointing towards $\langle ?x, ?y \rangle$ but not actively tracking the target and LOCKING$(?x, ?y)$ when the satellite is locking onto $\langle ?x, ?y \rangle$. The attitude is actively maintained only during the lock status. Once the images have been acquired the lock is released and re-acquired when a different area has to be observed. Hence transitions of the state variable ATT follows the loop $\rightarrow$ LOCKED$(\ldots) \rightarrow$ UNLOCKED$(\ldots) \rightarrow$ LOCKING$(\ldots) \rightarrow$ LOCKED$(\ldots) \rightarrow$ and so on.

The *Memory* MEM is modeled as resource timeline that keeps track of the spacecraft memory allocation. The GPS coordinates are stored in the timeline of a state variable GPS. This timeline is not under the control of the planner because it depends on the orbit of the satellite and cannot be controlled since OPS-SAT settings do not allow experiments to change the flight dynamics. It is generated in advance by quering an on-board service that gives information on when OPS-SAT will be covering a range of GPS coordinates and provided to the planner in the planning problem with goals to achieve. Since we are not interested in all the coordinates covered by the satellite along its orbit, we model a state variable taking only the following values: NOTINPOSITION(), denoting that the spacecraft is currently not covering any area of interest for the current planning session, and POSITION$(?x, ?y)$ when the satellite is above an area covering the $\langle ?x, ?y \rangle$ GPS coordinates.

The image classification sub-system is modeled by means of a state variable CLASS taking the following values (see Figure 5):

- CLASSIDLE(), when no classification algorithm is classifying at the moment;
- CLASSIFYCLOUDS$(?file_id, ?class)$, when the image taken by the camera with id $file\_id$ is currently being classified as cloudy or clear. The result is stored into the variable $?class$, that can take values CLEAR or CLOUDY.
- SAVEIMAGE$(?file_id)$ denotes the procedure to save the image once its been classified as clear.
- CLASSIFYINTEREST$(?file_id, ?target, ?x_t, ?y_t)$ when the image gets classified again to analyze the level of interest in the picture. If an object is found its type is stored in $?target$ and its coordinates in the picture are stored in $?x_t$ and $?y_t$.

The high level goal to take a picture can be achieved by having the camera component in the proper status to take a picture. The camera can take pictures when the attitude is locked onto the target while the lock status can be achieved only when the coordinates are visible. The plan is generated for *nominal* cases: for instance in case of a picture to be stored and classified, the picture is planned as clear, it is stored and no interesting object is found in the picture when classified. An example of nominal plan is shown in Listing 1.

If during execution the nominal plan is invalidated, a

Listing 1. "Nominal Plan"

```
  P |             MT               |      GPS        |       ATT        |      CAM        |      CLASS                           |        BAT          |
--------------------------------------------------------------------------------------------------------------------------------------------------------|
  0 | MTIdle()                     | NotVisible()    | Unlocked()       | CamIdle()       | ClassIdle()                          | [132.0,0.0]         |
  1 | TakePicture(10,51,id_0,T,T)  |                 |                  |                 |                                      |                     |
  2 |                              | Visible(10,51)  |                  |                 |                                      |                     |
  3 |                              |                 | Locking(10,51)   |                 |                                      |                     |
  4 |                              |                 | Locked(10,51)    |                 |                                      |                     |
  5 |                              |                 |                  | MakePic(id_0)   |                                      |                     |
  6 |                              |                 |                  |                 | ClassifyClouds(id_0,clear)           |                     |
  7 |                              |                 |                  |                 | SaveImage(id_0)                      | [132.0,132.0,18.0]  |
  8 |                              |                 |                  |                 | ClassifyInterest(id_0,none,-1,-1 )   | [222.0,222.0,0.0]   |
  9 |                              |                 |                  |                 | ClassIdle()                          |                     |
 10 |                              |                 |                  | CamIdle()       |                                      |                     |
 11 | MTIdle()                     |                 |                  |                 |                                      |                     |
 12 |                              |                 | Unlocked()       |                 |                                      |                     |
 13 |                              | NotVisible()    |                  |                 |                                      |                     |
 14 | <UNDECIDED>                  | <UNDECIDED>     | <UNDECIDED>      | <UNDECIDED>     | <UNDECIDED>                          | [222.0,222.0,0.0]   |
```

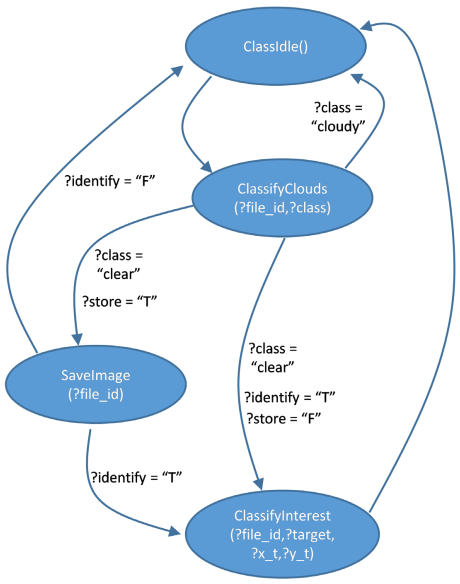

Figure 5. Classification State Variable

replanning step occurs. As an example, if when executing the plan in Listing 1 an image is classified as cloudy, a status `ClassifyClouds(id_0,cloudy)` is injected into the timeline. This status conflicts with the planned `ClassifyClouds(id_0,clear)`, fostering a re-planning from this status. In this case, according with the state machine in Figure 5, the saving and classification image procedures are discarded. As a consequence the memory occupation is also recalculated.

Regarding controllability, most of the states defined above can be executed under the control of the planner but have an uncertain duration. For these states, only bounds of acceptable duration can be decided at planning time, hence they are modeled with controllable starting point and contingent ending points. For instance the already mentioned locking value for the attitude subsystem or the take image value, whose actual duration depends on the acquisition camera procedures. Also the actual footprint in memory of an image can only be evaluated with some uncertainty. For this reason the flexible planning and the execution strategies mentioned in Section 3.1 and 3.3 are a necessary support for this experiment.

5. CONCLUSIONS

As its main contribution, the paper presents an on-board autonomous control architecture which combines a domain independent planner, a platform-agnostic executive, and an on-board image classification system. The project here described constitutes a first attempt to the application of Artificial Intelligence in ESA for on board autonomy on a flying mission. Initial steps were taken in the past to validate, on the ground segment, building blocks technologies such as diagnostics and automated planning.

In order to validate the architecture, a testing scenario is currently under the System Validation Test. The paper presents and discuss in detail a part of this test. The experiment uses the Nanosat Mission Operation Framework [9]. The NMF and the Experiment are implemented in Java 8. To run the experiment on the computer, we use the Netbeans IDE 8.2 and to simulate the satellite we use the Nanosat-MO-Simulator (Developer Version 2.0) of the NMF SDK. The Dell Latitude computer we use for this experiment is equipped with Windows 10 64-Bit, 16GB DDR4 SDRAM and an Intel Core i7-7600U with 2 cores and 2.89GHz processing speed. To test the experiment on real hardware, we use a MitySOM 5CSx System on Module dual-core 800MHz ARM Processor running with a Linux distribution named ngstrm 32-Bit.

Future work foresees to support the on-board system with a ground system that could complement and support the autonomous capability. The ground system will have the role of validating the plan generated on-board with respect to the overall mission objectives. For instance, the satellite can run multiple experiments in parallel and another experiment could require the use of the same payload (the on-board camera). For what concerns the extension of the on board capabilities, analysis of radio frequencies and magnetic forces as well as anomaly detection of telemetry values [15] is currently work in progress.

REFERENCES

1. James Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2. Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3. APSI. APSI Software Distribution Web Site. https://essr.esa.int/project/apsi-advanced-planning-and-scheduling-initiative, 2017.

4. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

5. A. Ceballos, S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. van Winnendael. A Goal-Oriented Autonomous Controller for Space Exploration. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, 2011.

6. S. Chien, M. Johnston, J. Frank, M. Giuliano, A. Kavelaars, C. Lenzen, and N. Policella. A Generalized Timeline Representation, Services, and Interface for Automating Space Mission Operations. In *Proceedings of the 12th International Conference on Space Operations, SpaceOps*. AIAA, 2012.

7. Steve Chien, Joshua Doubleday, David R Thompson, Kiri L Wagstaff, John Bellardo, Craig Francis, Eric Baumgarten, Austin Williams, Edmund Yee, Eric Stanton, et al. Onboard Autonomy on the Intelligent Payload EXperiment CubeSat Mission. *Journal of Aerospace Information Systems*, pages 1–9, 2016.

8. Steve Chien, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Rebecca Castano, Ashley Davis, Dan Mandl, Stuart Frye, Bruce Trout, and Seth Shulman. Using Autonomy Flight Software to Improve Science Return on Earth Observing One. *Journal of Aerospace Computing, Information, and Communication*, 2(4):196–216, 2005.

9. Cesar Coelho, Mario Merri, Otto Koudelka, and Mehran Sarkarati. OPS-SAT Experiments Software Management with the NanoSat MO Framework. In *AIAA SPACE 2016*, 2016.

10. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, May 1991.

11. J. Frank and A. Jonsson. Constraint Based Attribute and Interval Planning. *Journal of Constraints*, 8(4):339–364, 2003.

12. S. Fratini, F. Pecora, and A. Cesta. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.

13. P. Grandjean, T. Pesquet, A. M. M. Muxi, and M. C. Charmeau. What on-board autonomy means for ground operations: An Autonomy Demonstrator conceptual design. In *SpaceOps 2004*, 2004.

14. D. Lu and Q. Weng. A survey of image classification methods and techniques for improving classification performance. *International Journal of Remote Sensing*, 28(5):823–870, 2007.

15. J. Martinez Heras and A. Donati. Enhanced Telemetry Monitoring with Novelty Detection. *AI Magazine*, 2014.

16. Paul Morris. Dynamic controllability and dispatchability relationships. In *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings*, pages 464–479, 2014.

17. N. Muscettola. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., editor, *Intelligent Scheduling*. Morgan Kauffmann, 1994.

18. Nicola Muscettola, Paul H. Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998.*, pages 444–452, 1998.

19. Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

20. T. Nogueira, S. Fratini, and K. Schilling. Planning and Execution to Support Goal-based Operations for NetSat: a Study. In *Proceedings of the 10th International Workshop in Planning and Scheduling for Space (IWPSS-2017)*, 2017.

21. J. Ocon, J. Delfa, T. De la Rosa Turbides, A. Garca-Olaya, and Y. Escudero Martn. GOTCHA: An autonomous controller for the space domain. In *ASTRA-17. 14th Symposium on Advanced Space Technologies in Robotics and Automation*, 2017.

22. OPS-SAT. OPS-SAT Web Site. https://www.esa.int/Our_Activities/Operations/OPS-SAT, 2018.