Acta Futura 5 (2012) 53-63
DOI: 10.2420/AF05.2012.53

Acta Futura

Towards Robust Execution of Mission Plans for Planetary Rovers

Roberto Micalizio*, Enrico Scala, Pietro Torasso

Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino (Italy)

Abstract. The paper discusses a novel methodology for robust plan execution involving an intelligent agent called *Active Supervisor* (ActS). ActS aims at preventing (at least in some cases) the failure of durative actions by anticipating anomalous trends of execution and by properly handling them. To reach this result, ActS needs that the plan to be executed carries some important pieces of knowledge: besides preconditions and effects, actions must also be described by intermediate conditions (i.e., invariant conditions), which must be satisfied during the whole execution of durative actions. This knowledge is used by ActS to detect anomalous situations that may endanger the safeness of the plan executor. Whenever an anomaly has been detected, ActS tries to prevent a failure by changing the *execution modality* of the current action while it is still in progress. Preliminary experimental results, obtained in a simulated space exploration scenario, are reported.

1 Introduction

Planning the activities of a mobile robot is a complex task which many planners address by means of abstract models of the real world. In recent years there has been a substantial amount of work for reducing the gap between these abstract models and the real world, and innovative planning techniques have been proposed in literature. Nevertheless, *robust plan execution* still remains an open problem when it comes to deal with real-world environments that are just partially observable and not completely known in advance.

Some recent methodologies [2, 3, 7] face the problem of plan execution by establishing a closed loop of control including (among others) *on-line monitoring*, which is responsible for the detection of action failures, and *plan repair*, which is typically based on a re-planning step and it is devoted to restore (if possible) the nominal execution conditions. These methodologies, however, are unable to intervene during the execution of an action as the repair is invoked once an action failure actually occurs, and hence when the plan execution has been interrupted.

The space exploration scenario is a good example of challenging environments where innovative techniques for assuring robust and safe execution are required. In space exploration, however, there are many reasons why it may be difficult to adopt control methodologies just based on re-planning. First of all, the computational power onboard a planetary rover is in general insufficient to solve a complex task such as the synthesis of a new mission plan. More important, in this scenario a mission plan is the result of a long process involving a team of human experts: engineers and scientists have to cooperate to build a plan which achieves relevant scien-

*Corresponding author. E-mail: micalizio@di.unito.it

tific targets without endangering the rover integrity by taking into account resource consumption and physical limits; any change to the mission plan is therefore negotiated among team members. Since the synthesis of a mission plan is a complex task, the rover is typically not allowed to significantly deviate from it; thus, in case of action failures, the rover can just interrupt the execution and wait for instructions (i.e., a new mission plan) from a Ground Control Station (GCS) on Earth.

In this paper we propose to endow the rover with a controlling agent, called *Active Supervisor* (ActS), which aims at avoiding (at least in some cases) the occurrence of action failures improving the robustness of the plan execution phase. To reach this goal, ActS is able to adapt the way in which an action is carried on to the current contextual conditions; more precisely, each action in the plan is associated with a set of *execution modalities* which, similarly to [4], represent alternative ways to reach the action's effects. Intuitively, an execution modality can be seen as a specific configuration of the rover's devices. When an action is submitted for the execution, ActS is in charge of establishing, according to the current context [4], the initial execution modality of that action. Such an initial modality, however, can be adjusted online by ActS while the action is still in progress in order to adjust the rover's behavior without interrupting the execution phase. ActS is a complex system involving a number of modules; in particular, it exploits a temporal interpretation module for monitoring the execution of the mission plan and for detecting potential deviations from the nominal expected behavior over a time window. When potentially hazardous situations are detected, another module, called Active Controller, can decide to adjust the current execution modality by taking into account the suitability of alternative modalities in alleviating the discrepancy between the actual behavior and the nominal one.

The paper is organized as follows: section 2 describes a space exploration scenario used to exemplify the proposed approach; section 3 points out how a mission plan can be enriched to support ActS; section 4 focuses on ActS and describes both its internal architecture and the main strategies it follows. Finally, in sections 5 and 6 we discuss some preliminary experimental results and draw some conclusions.

2 A Space Exploration Scenario

This section introduces a space exploration scenario where a planetary rover is in charge of accomplishing explorative tasks. This scenario presents some interesting and challenging characteristics; the rover, in fact, has to operate in a hazardous and not fully observable environment where a number of unpredictable events may occur.

In our scenario the rover is provided with a mission plan covering a number of scientifically interesting sites. Such a plan requires to navigate from a location to another one, and involves a number of exploratory actions to be performed once a target has been reached; for instance we consider that the rover can:

- drill the surface of rocks;
- collect soil samples and complete experiments in search for organic traces;
- take pictures of the sites.

All these actions produce a certain amount of data which are stored in an on-board memory. Once a communication window towards Earth becomes available, the acquired data can be uploaded. In this paper, however, we do not consider the problem of communicating data in the space scenario, possible solutions tackling it are discussed in see [8, 10].

Figure 1 sketches an example of daily mission plan the rover is assigned with [in the map, altitude is represented in greyscale: white corresponds to the highest altitude, and black to the lowest]. It is easy to see that some of the plan actions can be considered as atomic (e.g., take picture), some others, instead, will take some time to be completed. For instance, a navigate action will take several minutes (or hours), and during its execution the rover moves over a rough terrain with holes, rocks, slopes. The safeness of the rover could be threatened by too deep holes or too steep slopes as the rover has some physical limits that cannot be exceeded; when this happens, the rover is unable to complete the action: the plan execution phase is stopped and a request for a recovery plan is sent to Earth. Of course, the rover's physical limits are taken into account during the synthesis of the mission plan, and regions presenting potential threats are excluded *a priori*.

The safeness of the rover, however, could also be threatened by terrain characteristics which can hardly be anticipated. For instance, a terrain full of shallow holes may cause undesired vibrations that may damage some rover's devices when prolonged over time. This kind of threat is difficult to anticipate from Earth; in fact, satel-

 DOI: 10.2420/AF05.2012.53

lite maps cannot capture all terrain details; moreover, this threat depends on the rover's contextual conditions such as its speed.

In the rest of the paper we suggest to endow the rover with a controlling intelligent agent, called Active Supervisor (ActS), which has to recognize potential anomalous trends while actions are still under way, and reacts to them in order to prevent the failure of the actions. To reach this goal, ActS is able to adjust the current *execution modality* of the action in progress so that the anomalous trend can be compensated and the nominal behavior is restored. For instance, the navigation action can be associated with three execution modalities: cruise-speed (the nominal one), high-speed and reduced-speed. In the following, we will show how the exploitation of these modalities allows ActS to flexibly respond to the contextual conditions. On the one hand, ActS can reduce the rover's speed in order to mitigate the harmful effects of a rough terrain; on the other hand, ActS can restore the cruise-speed (or even set the high-speed) when the terrain conditions get better.

3 Augmenting the Mission Plan

Before discussing the internal architecture of ActS and how it intervenes during the plan execution phase, it is essential to introduce the pieces of knowledge ActS requires to carry on its job. The first issue to face is about the representation of the mission plan. In this paper we assume that the plan has been synthesized on Earth with the help of automatic planners under the supervision of human experts. Since most of the automatic planners are based on PDDL (Planning Domain Definition Language which is *de facto* a standard for the representation of planning problems and domains), we also adopt this language to encode the high-level mission plan; in particular, we first point out advantages and limitations of PDDL, and then we present a number of extensions exploited by the active supervision mechanism we propose.

Durative actions in PDDL As a first approximation we can assume that the mission plan $\mathtt{P}$ assigned to a planetary rover is modeled as a sequence of actions $\mathtt{P} : \{\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_n\}$, each of which is an instance of a PDDL action schema. While some of these actions can be assumed instantaneous (e.g., taking a picture), some others are indeed *durative actions* (e.g., navigating along a path), supported by PDDL since version 2.1 [6]. Besides preconditions and effects, a durative action is also modeled in PDDL2.1 by a set of *invariant conditions*, which are required to hold over the time interval between the start and the end of the action. While this extension is adequate to solve a planning problem when actions cannot be assumed atomic, it is not sufficient to support our goal of detecting anomalous trends of executions; let us consider the following example.

Figure 2 shows a possible template for the `navigate` action in our space exploration scenario. In this case, the `condition` clause encodes the physical limits within which the rover's behavior is considered safe while a navigate action is in progress. Let us suppose that two parameters are essential for the safeness of a navigate action: `roll` and `pitch`. When either the absolute, or the derivate value of one of these two parameters exceeds a predefined threshold, the navigate action must be considered failed. In that case, ActS aborts the action in order to prevent further damages.

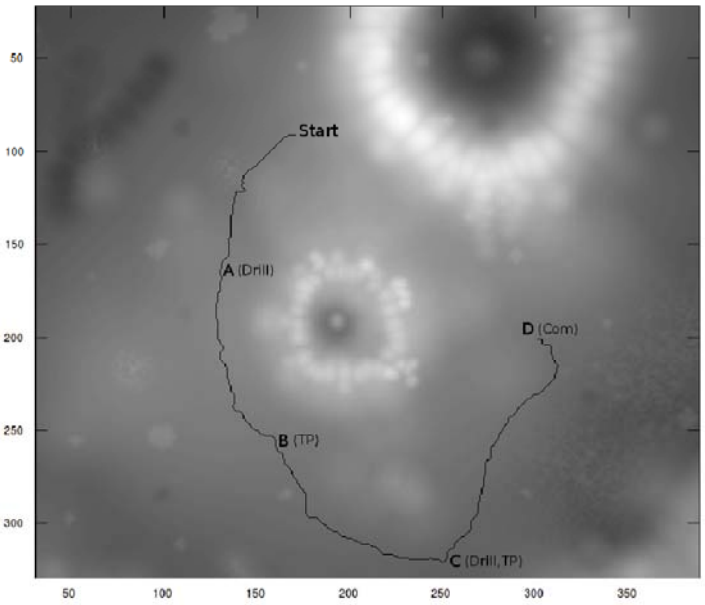
From the previous example it is apparent that an invariant condition is a Boolean statement, and represents a failure situation. When it is violated at some step of execution, it is too late for intervening: the plan execution phase must be interrupted and a new plan must be requested from Earth. On the contrary, we aim at preventing the occurrence of action failures by anticipating the violation of invariant conditions during the action execution. To do so we need some further pieces of knowledge associated with each action in the mission plan.

Execution Modalities The first extension we propose is to associate each action instance $\mathtt{a} \in \mathtt{P}$ with a set of *execution modalities*. Intuitively, an execution modality can be seen as a specific configuration of parameters or devices that has an impact on the current behavior of the rover. In other words, given the set of modalities $\mathtt{mods}(\mathtt{a})$ associated with $\mathtt{a}$, each execution modality $\mathtt{m} \in \mathtt{mods}(\mathtt{a})$ represents a possible way to reach the action's nominal effects $\mathtt{eff}(\mathtt{a})$.

For instance, the navigate action can be associated with three alternative modalities: `mods(navigate)`={*cruise-speed*, *reduced-speed*, *high-speed*}; *cruises-speed* is the nominal speed of the rover, while *reduced-speed* and *high-speed* are alternative modalities obtained by decreasing and increasing, respectively, the nominal speed.

This means that, during the execution of a navigate action, ActS can adjust the rover's velocity; this is important as the derivate values of `roll` and `pitch` will strongly depend on the actual speed of the rover; by tuning the speed parameter, ActS can therefore prevent

DOI: 10.2420/AF05.2012.53



An istance of daily mission plan:
- `navigate(Start,A);`
- `drill(A); navigate(A,B);`
- `take-picture(B);`
- `navigate(B,C); drill(C);`
- `take-picture(C);`
- `navigate(C,D);`
- `communicate(D).`

Capital letters A, B, C, D denote the sites the rover has to visit. Each site is tagged with the actions to be performed when the site has been reached: DRILL refers to the drill action, TP to take picture, and COM to data transmission. More actions can be done at the same site, see for instance target C. The black line connecting two targets is the route, predicted during a path planning phase, the rover should follow during a navigate action.

Figure 1. *An example of daily mission plan.*

```
(:durative-action navigate
parameters : (?r - rover ?y - site ?z - site)
duration : (= ?duration navigation-time ?z ?p)
condition : (and(at start(at ?r ?y)
        (over all (and(⩽ (pitch-derivate ?r) 5)
                (⩽ (roll-derivate ?r) 5)
                (⩽ (pitch ?r) 30)
                (⩽ (roll ?r) 30))))
effect :(and(at start(not (at ?r ?y)))
                (at end(at ?r ?z)))
```

Figure 2. *An example of durative action*

a failure of a navigate action.

Temporal Patterns of Behavior To prevent the occurrence of an action failure, it is essential to recognize anomalous trends of execution while the action is still in progress. Each action is therefore associated with a set of temporal patterns, each of which describes a sequence of events that should, or should not, occur during the nominal execution of an action. In this paper we advocate the adoption of the chronicles formalism [5] for encoding these temporal patterns. Intuitively, a chronicle is a set of events linked with each other by temporal constraints. Since they allow to model the behavior of dynamic systems over time, chronicles have been successfully exploited for of real-time monitoring and diagnosis of Discrete Event Systems (DES), even of large dimensions as telecommunication networks (see e.g., [11]).

In our case, the events that we want to capture within a chronicle correspond to relevant changes in the status of the rover which indicate potentially anomalous behaviors. These events may depend both on the activities carried on by the rover itself, and on the contextual conditions of the environment where the rover is operating. Thus, each durative action $a \in P$ is also associated with a set $\mathtt{chrons}(a) = \{\mathtt{chr}_1, \mathtt{chr}_2, \ldots, \mathtt{chr}_m\}$ of chronicles, where each $\mathtt{chr}_i$ represents a trend of execution, either nominal or anomalous, that is relevant for recognizing what is happening during the execution of action a.

Figures 3 and 4 show two examples of chronicle associated with a navigate action. The first chronicle indentifies a potentially hazardous terrain. This chronicle is recognized when at least N *severe-hazard* events have been detected within an interval of W time instants. The basic idea is that the safeness of the rover may be endangered when it moves at a high speed along a too rough terrain; this kind of threat can be captured by detecting hazardous variations of the roll and pitch parameters in a short time window. The second chronicle recognizes a plain terrain when for a period of at least N time instants the rover status remains nominal.

It is important to note that, to keep the definition of chronicles a simple task, they mention high-level events such as *medium-hazard* and *no-hazard*; these events re-

 DOI: 10.2420/AF05.2012.53

```
chronicle hazardous-terrain {
 event(medium-hazard[pitch, roll], t1 )
 event(medium-hazard[pitch, roll], t2 )
 event(severe-hazard[pitch, roll], t3)
 t1<t2<t3 ; t2-t1<W1 ; t3-t2<W1
 when recognized { emit
   event(hazardous-terrain[pitch,roll],t)} }
```

FIGURE 3. *A chronicle recognizing a hazardous terrain.*

```
chronicle plain-terrain {
  occurs( (N, +oo), no-hazard[pitch, roll], (t, t+W) )
  when recognized {
    emit event(plain-terrain[pitch,roll],t);
  } }
```

FIGURE 4. *A chronicle recognizing a plain terrain.*

sult from an interpretation process over the rover's status variables. In particular, it is easy to see that, since the execution of a navigate action is aborted when the rover's roll and pitch exceed predefined thresholds, these two parameters play an important role during the process of generating the high-level events mentioned within the chronicles in $\mathtt{chrons}(\mathtt{navigate})$. We will describe the interpretation process in the next section.

Execution Trajectories The last extension we introduce can be seen as a more fine-grained action model than a PDDL2.1 action schema. In the PDDL2.1 model, in fact, one just specifies (propositional) preconditions and effects, but there may be different ways to achieve the expected effects from the given preconditions. For instance, the model for action $\mathtt{navigate}(A, B)$ just specifies that: 1) the rover must be initially located in A and 2) the rover, after the completion of the navigate action, will eventually be located in B; but nothing is specified about the intermediate rover positions between A and B. This lack of knowledge may be an issue when we consider the problem of plan execution monitoring. For this reason, we allow to associate a durative action a with a parameter $\mathtt{trj}(a)$, that specifies a trajectory of nominal rover states. More formally, $\mathtt{trj}(a)=\{s_0, \ldots, s_n\}$, where s_i $(i : 0..n)$ are, possibly partial, rover states at different steps of execution of a. We just require that both $s_0 \vdash \mathtt{pre}(a)$ and $s_n \vdash \mathtt{eff}(a)$ must hold. Therefore, $\mathtt{trj}(a)$ represents how the rover status should evolve over time while it is performing a.

For instance, the trajectory of a navigate action maintains a sequence of waypoints, sketching the route the rover has to follow, which can be determined after a path planning process.

An execution trajectory can be very useful to determine deviations from the nominal execution that cannot easily detected by means of chronicles. For instance, it may happen that the rover is diverging from its target; however, since it is moving on a plain terrain, no erroneous chronicle is recognized as the rover is not endangered. Having an execution trajectory, in this case, could help ActS in deciding whether a deviation from the ideal path is the signal that something wrong has occurred.

Since it is not always possible to anticipate a possible trajectory for a given action, we consider an execution trajectory as an optional extension: if it is not available, ActS will just rely on chronicles and modalities.

4 Active Supervisor

In this section we present our Active Supervisor (ActS); in particular, we describe both its internal architecture and the algorithms which are at the basis of its inferences.

4.1 ActS's Architecture

Let us start with the ActS's internal architecture showed in Figure 5. ActS is a deliberative agent which directly interacts with the rover's Functional Layer (FL) (see e.g., [1, 4]). Since we want to emphasize the role of ActS in preventing action failures, in this paper we assume that ActS is not endowed with the ability of (re-) planning; therefore, the mission plan, augmented with the three extensions discussed above, must be an input from Earth.

DOI: 10.2420/AF05.2012.53

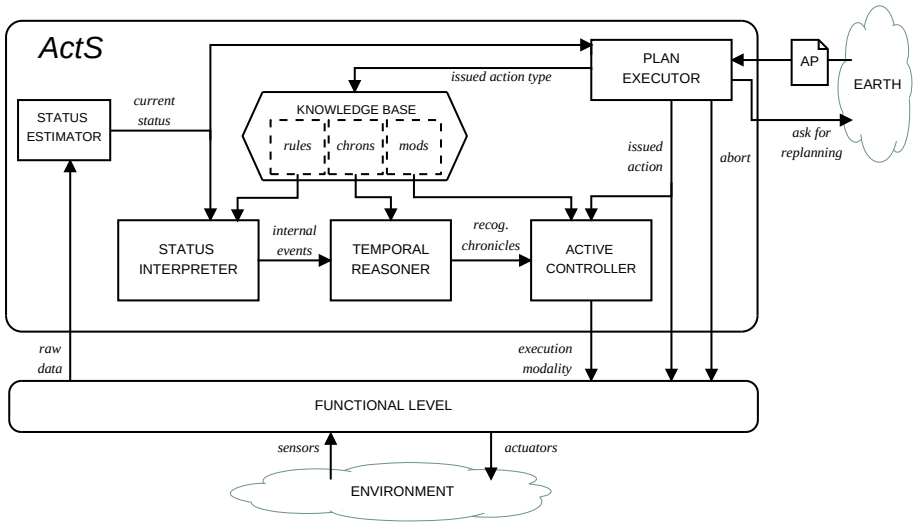


Figure 5. *ActS: internal architecture.*

The deliberative capabilities of ActS mainly regard the actual execution of the mission plan: ActS has to decide the initial execution modality of each action included in the plan and, once the action has been submitted to the FL, it is in charge to correct such a modality when observations from the environment suggest that something wrong is occurring.

The internal architecture of ActS involves a number of modules:

- The **Status Estimator** (SE) gets, at each time t, the raw data provided by the FL and produces an internal representation of the current rover's status. The result of the SE can be thought of as an array $\mathtt{status}_t$ of status variables representing a rover's snapshot at time t. The SE can carry out qualitative abstractions of the collected data, therefore each variable $v \in \mathtt{status}_t$ is set to a specific value which may be either a quantitative or qualitative. The internal status $\mathtt{status}_t$ produced by the SE is made available to the Plan Executor and to the Status Interpreter - two other modules of ActS.

- The **Plan Executor** (PE) is responsible for the actual execution of the actions in the given mission plan. This means that the PE must be able to determine: firstly, whether the current action to perform can be submitted for execution to the FL (i.e., when the action preconditions are satisfied in $\mathtt{status}_t$), and secondly, when an action has been carried out successfully. Whenever the PE detects an anomalous condition (e.g., an action takes longer than expected) it aborts the execution (by sending an appropriate abort command to the FL) and asks for a new repair plan.

- The **Knowledge-Base** (KB) maintains the pieces of knowledge exploited by ActS for detecting anomalous trends and for compensating them by tuning the execution modality of the current action. In particular, for each action type $\mathtt{type}$, the KB maintains the set of execution modalities $\mathtt{mods}(\mathtt{type})$ and chronicles $\mathtt{chrons}(\mathtt{type})$ associated with it. Moreover, the KB includes a set of interpretative rules that help the Status Interpreter in its job (see later).

- The **Status Interpreter** (SI) has to generate the internal, high-level events that are mentioned within the chronicles. It exploits a set of interpretative rules associated with the current action. These interpretative rules have the form *Boolean condition* $\rightarrow$ *internal event*. The Boolean condition is built upon three basic types of atoms: status variables x_i, status variable derivates $\delta(x_i)$, and abstraction operators $\mathtt{qAbs}(x_i, [t_l, t_u]) \rightarrow \mathtt{qVals}$ which map the array of values assumed by x_i over the time interval $[t_l, t_u]$ into a set of qualitative values $\mathtt{qVals} = \{\mathtt{qval}_1, \ldots, \mathtt{qval}_m\}$. For example, the following interpretative rules:

  $(\delta(\mathtt{roll}) > \mathtt{limits}_{\mathtt{roll}} \vee$
  $\delta(\mathtt{pitch}) > \mathtt{limits}_{\mathtt{pitch}})$

 DOI: 10.2420/AF05.2012.53

$\rightarrow$ severe-hazard(roll, pitch)

are used to generate a severe-hazard event whenever the derivate value of either roll or pitch exceeds predefined thresholds in the current rover status.

Another example is the rule:

attitude(roll, $[t_{current} - \Delta, t_{current}]$) = nominal $\wedge$
attitude(pitch, $[t_{current} - \Delta, t_{current}]$)= nominal
$\rightarrow$ safe(roll, pitch)

where attitude is an operator which abstracts the last Δ values of either roll or pitch (the only two variables for which this operator is defined) over the set {nominal, border, non-nominal}.

- The **Temporal Reasoner** (TR) is essentially a Chronicle Recognition System (CRS) similar to the one proposed by Dusson in [5]. It is responsible for triggering the Active Controller once a new chronicle has been recognized.

- The **Active Controller** (AC) accomplishes two important activities. First, it selects an execution modality to be issued towards the FL. In principle, such a selection should correct the current robot's behavior smoothly; that is, on one side, the AC's strategy should not be too reactive in order to avoid abrupt changes in the robot's behavior which may be as dangerous as the threat to face; on the other side, the AC should be able to restore the nominal execution modalities when it is reasonable to presume that no menace is expected in the near future. Second, the AC updates some parameters of the current action according the execution modalities it emits. For instance, when a navigation action is slowed down, it will take more time to be completed, this extra time must be taken into account by the PE during its job.

4.2 Algorithms

In the previous subsection we have introduced the main modules of ActS; in this subsection we discuss how these modules are actually integrated with one another to provide the supervision service ActS is responsible for. In particular, we describe two main phases: one regarding the high-level control loop (which determines when a new action can be submitted for the execution or when an action in progress must be aborted), and one regarding the active monitoring process, which is in charge of tuning the execution modality of the action currently in execution.

The main control strategy is sketched in Figure 6. As said above, this is the main control loop through which the execution of the mission plan is supervised. In particular, such a loop iterates over the time and involves two steps: 1) the selection of a new action to be submitted, and 2) the supervision of a (durative) action in execution.

The first phase (lines 06 through 11) is activated whenever the current action a becomes *null*; this means that the previous action has been successfully carried out (lines 16 and 17) and a new action has to be executed. Thus, the next action in the plan is extracted and its preconditions are assessed against $status_t$ (i.e., the current status estimated by the SE). If the action preconditions are not satisfied, the action is not executable; since ActS is not endowed with replanning capabilities, it responds to the anomalous situation by stopping the plan execution phase and by asking Earth for a new mission plan. On the contrary, if the action is executable in $status_t$, ActS has to decide the initial execution modality of a according to its action type and to the contextual conditions encoded by $status_t$ (this step is similar to the "context aware" modality selection in [4]). Note that ActS selects the initial execution modality for durative as well as atomic actions; of course, while ActS has also to adjust the modality of durative actions while they are in progress (second phase, see below), the granularity of atomic actions does not require further adjustments once they are sent for execution.

The second control phase (lines 12 through 19) is activated whenever a durative action is in progress. In this phase, ActS first looks for deviations between the nominal evolution of the action and the actual one. To this end, it checks three conditions: 1) assesses whether some invariant conditions associated with the action have been violated in $status_t$; 2) establishes the current duration of the action and compares it with the estimated one; 3) estimates the current execution trajectory and evaluates how far it deviates from the nominal one. If at least one of these three conditions is violated, ActS sends an abort command to the FL, interrupting the current action, and asks Earth for a new plan (line 14). On the contrary, no violation has been detected, and ActS assesses whether the action has reached its expected effects. This is done by comparing the nominal effects with the status estimation $status_t$: if they

DOI: 10.2420/AF05.2012.53

```
ActS::MainStrategy(P)
01 t ←0
02 a ←null
03while there are actions in P to be performed
04   rdata_t ← getRawDataFromFL(t)
05   status_t ← StatusEstimator(rdata_t)
06   if (a is null)                    // an action a has to be selected
07     a ←getNextAction(P)
08     if checkPreconditions(pre(a), status_t) = not satisfied
09         ask for replanning and exit
10     currentmod(a) ←selInitMod (status, acttype(a))
11     submit a to FL with modality currentmod(a)
12   else
13     if (checkInvariants(inv(a), status_t) = violation) ∨
       (actualDuration(a) > duration(a)) ∨
       (trajectoryDeviations(trj(a), status_t)=relevant)
14       send abort to FL
15       ask for replanning and exit
16     if hold(eff(a), status_t)      // a is completed with success
17       a ← null
18     else
19       ActiveMonitoring(a,t,status_t,currentmod_a)
20   t ← t + 1
```

Figure 6. *The Plan Executor's high-level control strategy.*

```
ActS::ActiveMonitoring(a, t, status_t,currentmod(a))
  H ← append(H, status_t)
  rules(a) ← get-interpretative-rules(KB, acttype(a))
  events_t ← StatusInterpreter(H, rules(a))
  RC ← ∅
  chronicles(a) ← get-chronicles(KB, acttype(a))
  for each event e_t ∈ events_t
      chr(a) ← get-relevant-chronicle(e_t, chronicles(a))
      if TemporalReasoner(chr(a), e_t) emits recognized
          RC ← RC ∪{chr(a)}
  if RC ≠ ∅
      mods(a) ←get-execution-modalities(KB,acttype(a),RC)
      newmod ←ActiveCntr(RC,mods(a),currentmod(a))
      currentmod(a) ← newmod
      submit currentmod_a to FL
```

Figure 7. *The strategy for the active monitoring.*

match, the action is completed (line 16) and the action pointer a is set to *null* so that the first phase is activated again.

When an action is neither aborted nor finished, it is still in progress; in that case ActS is in charge of tuning the execution modality of the action according to the potential anomalous trends that are recognized; this task is carried out by the *ActiveMonitoring* pseudo-function sketched in Figure 7.

The purpose of *ActiveMonitoring* is to emit (when necessary) an execution modality towards the FL so that the trend of the current action can be corrected. This result is obtained by the cooperation of three modules of ActS: *Status Interpreter* (SI), *Temporal Reasoner* (TR) and *Active Controller* (AC); the pseudo-code in Figure 7 describes how these modules are actually integrated.
Since the TR is based on chronicles, the first step to accomplish requires the generation of the internal events that, mentioned within the chronicles, will be consumed by TR. This task is up to the SI, which exploits a history of rover's past states together with a set of interpretative rules in order to synthesize the internal events representing relevant changes in the rover status (see the first three lines): $events_t$ is the set of internal events generated by the SI at each time t. Each event $e_t \in events_t$ is subsequently sent to the TR. For simplicity, in our approach we assume that each event e_t can be consumed by exactly one active chronicle chr; the function *get-relevant-chronicle* in Figure 7 selects such a chronicle from chronicles(a) so that the TR receives in input just the event e_t which influences that chronicle. By consuming the events it receives from the SI, the TR will eventually recognize a chronicle chr_a. In general, more chronicles can be recognized simultaneously, so all the chronicles recognized at time t are collected into the set RC, which becomes the input for the AC as well as the current execution modality. This last module has the responsibility of updating the current execution modality from the set of possibilities mods(a); the new modality will then be emitted towards the FL. In our current and preliminary solution, the AC receives just one chronicle at a time and matches that chronicle with a specific execution modality by exploiting a predefined set of rules.

4.3 Running example

To show the effectiveness of ActS, in this subsection we compare the execution of the rover's mission plan (introduced in section 2) focusing on a specific navigate action - navigate(A,B) - in two different situations: first when ActS is off, so no execution modality is adjusted on-line; second when ActS is on and is in charge of adjusting the execution modality of durative actions while they are in progress. Of course, in both situations the violation of invariant conditions causes the abortion of the current action. Figure 8 plots the derivate value of the roll parameter over the time, while the navigate action is in progress in the two situations: when ActS if off (above) and when ActS is on (below). It is apparent that, when ActS is off, the execution of the navigate action is stopped after a number of time instants (in our experiments we sampled the rover's status every second). This happens because of the violation of the invariant conditions associated to the navigate, which require that the derivate value of the roll parameter must be below 5 de-

 DOI: 10.2420/AF05.2012.53

grees. On the contrary, when ActS is on, the navigate action can be carried out successfully; in fact, ActS recognizes a chronicle `hazardous-terrain`, and intervenes at time instant 87 by setting the action modality to *reduced speed*: such a change has a positive effect on the roll derivate, which does not exceed the threshold, and therefore the navigate can go on until the final target (B) is reached. [1]

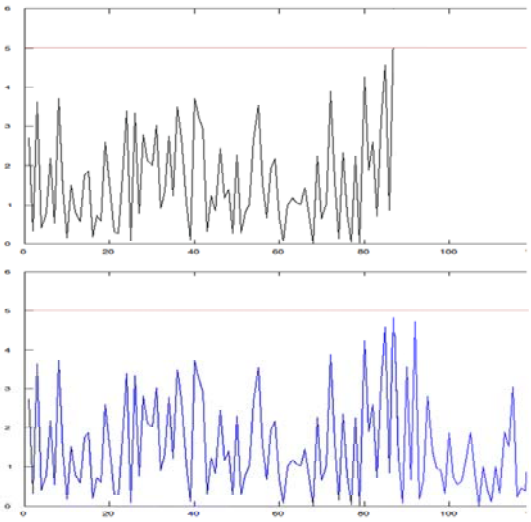


Figure 8. *The derivate of the roll parameter during action* `navigate(A, B)`*: when ActS is off (above) and when ActS is on (below).*

5 Experimental Results

The experimental scenario ActS has undergone to a first validation by using as test bed the space exploration scenario previously introduced. The planetary environment has been represented as a Digital Elevation Model (DEM); we assumed that an initial DEM D_{init} (presumably computed from satellites images), is available, and is used for implementing and experimenting a set of rover's missions. In particular, by taking into account the terrain's characteristics, we have subdivided the rover's missions into two classes: *easy* and *difficult*. Note that the planning phase verifies the feasibility of each navigate action by invoking a path planner that, relying on D_{init}, assesses the validity of the invariant conditions associated with this action type (see Figure 2) and provides also a trajectory in terms of way points.

Since D_{init} is just an approximation of the real terrain, the actual execution of a mission plan may be affected by unexpected environmental conditions. For simulating the discrepancies between D_{init} and the real terrain, we have altered the original DEM by adding a random noise on the altitude of each cell. In our experiments, we have considered 6 noise degrees: from 10 cm to 15 cm, and for each of them we have generated 320 cases: 160 for the *easy* class and 160 for the *difficult* one.

Altogether, in our experiments we have considered up to 1920 navigate actions differing with one another for their starting and ending points, and their length.

To prove the effectiveness of our control architecture, we have simulated the execution of both *easy* and *difficult* cases in each noised DEM comparing the rover's behavior both when ActS is off and when ActS is on. A simplified simulator of the FL has been implemented in order to generate with a frequency of 1Hz the set of raw data. For measuring the robustness of the plan execution and for providing some insights about the ability of ActS in tolerating variations in the DEM, we are reporting data about three main parameters concerning the execution of the *navigate* actions:
1) the percentage of navigate actions that were completed successfully.
2) the percentage of path actually covered by the rover with respect to the estimated trajectory.
3) The percentage of steps the navigation has been performed in the slowdown modality w.r.t. the number of steps in the estimated trajectory. (This datum is meaningful just when Acts is on.)
Figure 9 summarizes the results of the tests. The graphs show the average values for the class of difficult cases (solid line), and for the class of the easy ones (dashed line). Each bullet corresponds to the average value of 160 navigations; squares denote the responses when ActS is off, triangles denote the responses when ActS is on. It is easy to see that ActS-*on* always provides better results than ActS-*off* as concerns both the percentage of success and the progress. As expected, in the *difficult* cases, the gains are significant even for small DEM deviations, whereas in the *easy* ones, the gain becomes relevant when map deviations are more significant. The results also show that the ActS is quite powerful but cannot avoid failures when the noise degree grows too much.

A final remark concerns the cost of ActS: while the computational cost is negligible, there is an impact on the actual execution that we estimate as the percentage

[1] For the sake of discussion we only consider the roll parameter; however, the interpretation rules used to generate the internal events actually take into account a subset of parameters .

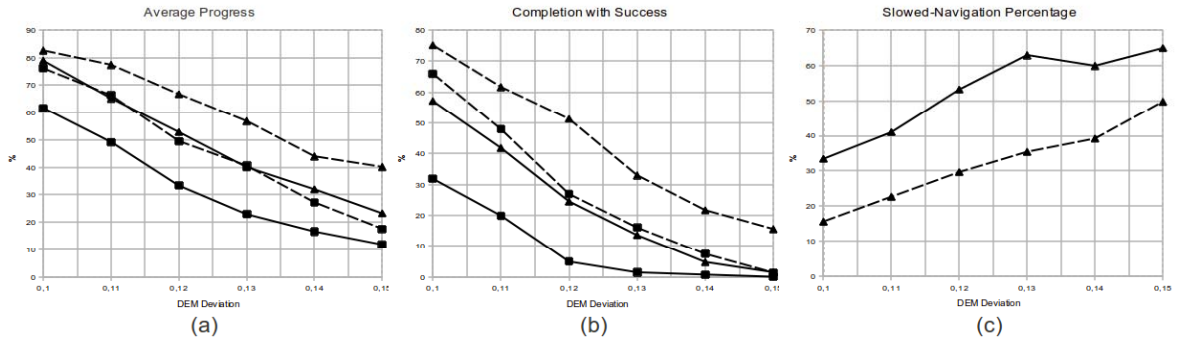


FIGURE 9. *Experimental results.*

of steps performed in the reduced-speed modality (see Figure 9.c): this percentage is proportional to the noise degree and represents and measure of the difficulty of completing the navigation.

Implementation. To implement ActS we have used PLEXIL and its environment (the Universal Executive). PLEXIL is a light weight [PLEXIL has been designed for very low performance systems, and it has been successfully tested on a VxWorks OS (see [13])] but powerful language developed by the NASA [12] [PLEXIL is downloadable at http://plexil.wiki.sourceforge.net], which provides a useful event driven framework supporting concurrent tasks. While PLEXIL has been exploited for the realization of the control strategies depicted in figures 6 and 7 (and hence for the coordination of the different internal modules of ActS), each internal module has been developed as a single piece of software in Java or in C++. The experiments have run on a laptop Intel Core2 2.16 GHz, 2 GB RAM.

6 Discussion and Conclusions

The paper has addressed the problem of robust plan execution when the environment may (slightly) differ from the one known (or assumed) during the planning phase and unexpected contingencies may arise. Previous works in literature have faced this problem by endowing the plan executor (i.e., a planetary rover in our case study) with some form of autonomous behavior. For instance, the control architectures discussed in [1, 4, 9] support the robot's autonomy by means of three layers of control: the highest one is devoted to the decisional aspects and it is typically based on one (or even more) (re)planning module(s). Recent works on planning under uncertainty have faced the problem of recovering from an action failure by synthesizing a repairing plan on the fly (e.g., see [7, 3, 2]). These approaches, however, have been designed to intervene only after a failure has occurred, and therefore when the plan execution has been interrupted.

In this paper we have tackled the robust plan execution problem from a different perspective. Rather than activating a re-planning phase after the detection of an action failure, we have proposed to endow the rover with an intelligent controller, called Active Supervisor (ActS), whose aim is to prevent action failures. In particular, we have discussed both the internal architecture of ActS and its main strategies. Moreover, in order to reach the goal of preventing failures, we have also discussed and motivated the necessity of augmenting a mission plan with further pieces of knowledge. Two extensions play a central role. First, the set of temporal patterns that are relevant for a given action type. These patterns describe the rover's behavior during the execution of a (durative) action and are used by ActS to identify anomalous trends. In the paper we have adopted the formalism of chronicles [5] to encode temporal patterns as it represents a viable and efficient solution to the problem of interpreting online the raw data coming from the environment, and hence reasoning about the environment in more abstract terms.

The second extension is about *execution modalities*, which associated with each action type, represent at an abstract level how ActS can intervene during the execution of an action. Different execution modalities correspond to different rover's settings, and hence to different evolutions of the action in progress. ActS can therefore decide to change the current execution modality of an

 DOI: 10.2420/AF05.2012.53

action in order to compensate an anomalous trend. The adoption of execution modalities is not completely new, also in [4] the authors suggest a methodology for adjusting the way in which an action is carried on depending on the rover's current context. However, their context is just a snapshot of the rover's status plus the status of environment surrounding the rover. Conversely, by means of the chronicle formalism, we are able to deal with a "temporal" context; which is much more informative than a single snapshot and allows us to predict with a higher confidence how the context will evolve in the immediate future. As a consequence, we can anticipate the change of modality.
As future work we intend to improve the ActS agent in two ways. First, ActS has to be more flexible in the selection of new execution modalities. In the current implementation, the selection of a new modality is up to the Active Controller, which exploits a set of predefined (and static) rules; to make this selection more flexible, also the changes made in the recent past should be considered.

Second, ActS has to change an execution modality not only on the basis of the safeness of the action currently in progress, but also on the basis of global constraints associated with the plan. For instance, in the hypothesis that some mission goals have to be reached within a time deadline, ActS has to estimate whether the deadline can be met, and in the negative case ActS can decide to increase the rover's speed.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research*, 17(4):315–337, 1998.

[2] A. Bouguerra, L. Karlsson, and A. Saffiotti. Monitoring the execution of robot plans using semantic knowledge. *Robotics and Autonomous Systems*, 56:942–954, 2008.

[3] M. Bozzano, A. Cimatti, M. Roveri, and A. Tchaltsev. A comprehensive approach to onboard autonomy verification and validation. In *ICAPS'09 Workshop on Verification and Validation of Planning and Scheduling Systems*, 2009.

[4] D. Calisi, L. Iocchi, D. Nardi, C. Scalzo, and V. A. Ziparo. Context-based design of robotic systems. *Robotics and Autonomous Systems (RAS) - Special Issue on Semantic Knowledge in Robotics*, 56(11):992–1003, 2008.

[5] C. Dousson and P. Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI'07*, pages 324–329, 2007.

[6] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[7] R. Micalizio. A distributed control loop for autonomous recovery in a multi-agent plan. In *Proc. IJCAI'09*, pages 1760–1765, 2009.

[8] I. Musso, R. Micalizio, E. Scala, et al. Communication scheduling and plans revision for planetary rovers. In *Proc. of i-SAIRAS'10*, 2010.

[9] I. Nesnas. Claraty: A collaborative software for advancing robotic technologies. In *Proc. of NASA Science and Technology Conference*, 2007.

[10] A. Oddi and N. Policella. Improving robustness of spacecraft downlink schedules. *In IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 37(5):887–896, 2007.

[11] Y. Pencolé and M. Cordier. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164:121–170, 2005.

[12] V. Verma, T. Estlin, A. Jònsson, C. Pasareanu, R. Simmons, and K. Tso. Plan execution interchange language (PLEXIL) for executable plans and command sequences. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, 2005.

[13] V. Verma, A. Jònsson, C. Pasareanu, and M. Iatauro. Universal executive and plexil: Engine and language for robust spacecraft control and operations. In *American Institute of Aeronautics and Astronautics Space Conference*, 2006.