Control of Autonomous Mobile Robots with Automated Planning

Ezequiel Quintero Ángel García-Olaya Daniel Borrajo Fernando Fernández
Departamento de Informática. Universidad Carlos III de Madrid
Av. de la Universidad, 30. Leganés (Madrid). Spain
equinter@inf.uc3m.es

***Abstract*—In this paper we present an approach for the control of autonomous robots, based on Automated Planning (AP) techniques, where a control architecture was developed (ROPEM: RObot Plan Execution with Monitoring). The proposed architecture is composed of a set of modules that integrates deliberation with a standard planner, execution, monitoring and replanning. We avoid robotic-device and platform dependency by using a low level control layer, implemented in the Player framework, separated from the high level task execution that depends on the domain we are working on; that way we also ensure reusability of the high and low level layers. As robot task execution is non-deterministic, we can not predict the result of performing a given action and for that reason we also use a module that supervises the execution and detects when we have reached the goals or an unexpected state. Separated from the execution, we included a planning module in charge of determining the actions that will let the robot achieve its high level goals. In order to test the performance of our contribution we conducted a set of experiments on the International Planning Competition (IPC) domain *Rovers*, with a real robot (Pioneer P3DX). We tested the planning/replanning capabilities of the ROPEM architecture with different controlled sources of uncertainty.**

***Index Terms*—Automated Planning, Autonomous Robots, Robotic Architectures, Mobile Robots.**

I. Introduction

INTELLIGENT agents, such as mobile robots, are used in dynamic environments that entail sensor noise, in addition to the uncertainty of task execution on the real world and the difficulty of environment modeling. This represents a major challenge when applying any control technique to robotics.

Many approaches have been presented so far to coordinate sensing and acting in robot control. Most previous work implements reasoning in robotic tasks as reactive systems, with very little deliberation. In our work we propose the use of Automated Planning (AP) [1] to implement the deliberative step between observation and action execution. The field of AP has a remarkable activity but most of its research is done on theoretical domains of great scientific interest, that are implemented in real life only in the solution of problems within specific contracts, with private or public organizations. This is due to the complexity and cost involved on reproducing the problems of scientific interest in reality. Nevertheless, nowadays the AP field is receiving much attention from various production sectors such as logistics [2], satellites [3], [4], critical and control decision systems [5], [6], and even military operations and evacuation [7].

The difficulty of applying AP to robot control arises when we have to generate an accurate description of the control tasks, which is essential for the planning process. We overcome this challenge by including supervision to the task execution. This provides us with replanning capabilities. By including monitoring we can generate new plans to reach the final goals when the initial plan fails. So, with our work we are proposing a solution that will allow us to partially deal with the environment uncertainty (which is a common problem to all robotic control approaches) and will also allow us to apply planning to real world problems.

One of the weaknesses of most robotic control systems is that they are commonly linked to specific robotic devices. In our approach we solve this problem by separating the robot-platform control from the high level deliberation. AP techniques could also be applied to low level control but in this research we focus on high level task planning. Monitoring could also be done at the low level but for now we are just focusing on high level.

In the planning step we use an environment model - dynamically created from the sensors (low level) information - and manage high level task execution. By translating the high/low level information we achieve the mentioned control level independence. And by making use of AP techniques we benefit from its standard definition language; obtaining domain, problem and planner independence.

Our approach was tested on a real robot (Pioneer P3DX) using the International Planning Competition (IPC[1]) domain *Rovers*. This domain is inspired on the Mars exploration rover missions, and allows us to represent a set of mobile robots that can traverse waypoints on a planet, collecting samples and sending data to a lander. Problems involve task and path planning.

The rest of the paper is organized as follows. In Section II we introduce automated planning. In that Section we also introduce the Rovers domain. In Section III we describe the ROPEM architecture that allowed us to carry out our contribution. In Section IV, we discuss the experiments we conducted in order to test the proposed approach. In Section V, the related work is presented. And finally, the conclusions and future work are summarized in Section VI.

[1]IPC: http://ipc.icaps-conference.org/

II. Automated Planning

On this work, we focus on the classical AP approach (also known as STRIPS planning) with action costs. A STRIPS planning problem with action costs can be defined as a tuple $P = \{F, A, I, G, c\}$, where:

- F is a finite set of grounded predicates and functions
- A is a finite set of actions, being each $a_i \in A$ composed of preconditions establishing when the action can be applied, and effects, consisting of elements of F being added or deleted from the current state after a_i is applied
- $I \subseteq F$ is the initial state, i.e. a subset of F that represents the set of grounded literals that are true at the start of the planning process
- $G \subseteq F$ is the set of goals, i.e. a subset of F that must be true for the problem to be solved, and
- c is a function $c : A \mapsto \mathbb{R}_0^+$ that defines the cost of each action

A solution of the planning problem P is an ordered list of actions $\Pi = \{a_0, a_1 ... a_n\} | a_i \in A$, which applied to the initial set of facts I results in a state where all the elements of G are true. The cost of the plan Π is defined as $C(\Pi) = \sum_{a_i \in \Pi} c(a_i)$.

One of the main advantages of AP is the availability of a standard representation language, *Planning Domain Definition Language* (PDDL [8]). PDDL permits us to represent domains (objects of different types, predicates and actions) and problems (initial state and goals), providing us with planner/domain independence and nowadays also allowing us to take into account action costs, state preferences, and action durations. But specification of accurate action models for addressing AP tasks in the real world, like robotic control, is complex. Current technology does not allow us to extract all the information about the environment, so our vision of the world through sensors can not be fully informed. Even in traditionally easy-to-code planning domains, it is complex to specify the potential outcomes of actions when the environment is non-deterministic. So, most of the time the success of the AP systems fully depends on the skills of the experts that define the action model. In the real world defining these models is particularly difficult because of the action execution uncertainty, especially in the environments where autonomous robots are used. Furthermore, due to the above, generated plans can not always be successfully completed in reality.

Deterministic planning does not seem useful for control systems by itself, but it can be improved to take advantage of the benefits it provides (like domain/planner independence, long-term reasoning and explicit representation of states, goals and actions). Real world difficulties can be overcome by supervising the execution and planning process, and the planning community is currently developing systems for the acquisition, validation and maintenance of AP models to improve the knowledge representation issues (like for instance, integrating planning and learning to improve execution [9]).

For non-deterministic environments, probabilistic planning techniques can be used. And there is also a standard representation language *Probabilistic Planning Domain Definition Language* (PPDDL [10]) for this planning approach. This language allows us to include information about the probabilities of different effects of action execution, allowing a more realistic representation in some domains. For now, we focus on classical planning and we deal with the uncertainty by adding monitoring. The reason is that the use of probabilities complicates the planning process and generating the probability models is difficult.

We tested the performance of our approach using the IPC domain *Rovers*. The Rovers domain is inspired on the control of planetary rovers. A set of rovers navigate a planet surface, finding samples and communicating this information back to a lander spacecraft. The domain includes the following objects: rovers, their stores, cameras, waypoints, soil/rock sample waypoints, objectives, and a lander-waypoint where the lander spacecraft is located.

Each rover is situated at a given location, and it can carry a sample of a given waypoint or be empty. Taken samples may or may not have been communicated to the lander. Each rover will be able to:

- Take images of an objective with the calibrated camera.
- Traverse the path between two connected waypoints.
- Load soil/rock samples of a waypoint into the store.
- Transmit data for a sample or image.
- Empty the store.

To illustrate the domain definition with PDDL we are now going to describe *Rover* domain specification. In Figure 1 we can see how to define a PDDL domain. In line 1 we indicate the domain name, in 2 we specify that it is a typed domain, and in lines 3 and 4 the types of objects present at the domain are listed. In the case of this domain, the object types are: rover, waypoint, camera, mode (for the modes that the camera allows), lander (for the lander spacecrafts) and objective (points to be photographed).

```
1 (define (domain Rover)
2    (:requirements :typing)
3    (:types rover store waypoint lander
4            camera mode objective)
5 ...
```

Fig. 1. Initial part of the Rovers domain PDDL definition.

In Figure 2 the predicates are declared, so we can represent the world states. For example, predicate `at` (line 2) is used to indicate the current location of the rover, `at-lander` (line 3) indicates the lander spacecraft waypoint, the predicates from lines 5 to 7 describe the rovers instruments (`equipped-for-xxx`) and the predicates between lines 17 and 19 describe the data communication objectives (`communicated-xxx-data`).

Figure 3 shows the definition of the `take-image` action. The parameters involved of the actions are declared in lines 2 and 3. The rover `?r` takes an image of the objective `?o` from the waypoint `?p`, with the camera `?i` on mode `?m`. Between lines 4 and 9, the preconditions of the action are detailed. The rover must be equipped for imaging (line 4), the objective has to be visible from the current waypoint (line 5), the camera involved has to be calibrated (line 6), be on the rover (line

```
(:predicates
   (at ?x - rover ?y - waypoint)
   (at_lander ?x - lander ?y - waypoint)
   (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
   (equipped_for_soil_analysis ?r - rover)
   (equipped_for_rock_analysis ?r - rover)
   (equipped_for_imaging ?r - rover)
   (empty ?s - store)
   (have_rock_analysis ?r - rover ?w - waypoint)
   (have_soil_analysis ?r - rover ?w - waypoint)
   (full ?s - store)
   (calibrated ?c - camera ?r - rover)
   (supports ?c - camera ?m - mode)
   (available ?r - rover)
   (visible ?w - waypoint ?p - waypoint)
   (have_image ?r - rover ?o - objective ?m - mode)
   (communicated_soil_data ?w - waypoint)
   (communicated_rock_data ?w - waypoint)
   (communicated_image_data ?o - objective ?m - mode)
   (at_soil_sample ?w - waypoint)
   (at_rock_sample ?w - waypoint)
   (visible_from ?o - objective ?w - waypoint)
   (store_of ?s - store ?r - rover)
   (calibration_target ?i - camera ?o - objective)
   (on_board ?i - camera ?r - rover)
   (channel_free ?l - lander)
)
```

Fig. 2. Rovers domain PDDL predicates.

7) and support the photography mode we want for the picture (line 8). Finally, the effects of the action are that we have the image (line 10) and that the camera is no longer calibrated (line 11).

```
(:action take_image
 :parameters (?r - rover ?p - waypoint ?o - objective
              ?i - camera ?m - mode)
 :precondition (and (equipped_for_imaging ?r)
                    (visible_from ?o ?p)
                    (calibrated ?i ?r)
                    (on_board ?i ?r)
                    (supports ?i ?m)
                    (at ?r ?p))
 :effect (and (have_image ?r ?o ?m)
              (not (calibrated ?i ?r))))
```

Fig. 3. Definition of the `take-image` action in PDDL language.

III. Architecture

In this section, the architecture that integrates planning, execution, monitoring and re-planning (ROPEM: RObot Plan Execution with Monitoring) is described. This is the first step of a long-term goal that consists on integrating other techniques for improving the autonomous robot control, refining the different modules presented in this section and adding new ones (such as several machine learning modules, that would merge learning reactive behaviours, with learning control and domain knowledge, for the deliberative planner).

In Figure 4 the system execution is described. ROPEM is initiated by loading a domain (line 3) and a problem (line 4). Right after that, we extract from the problem the information about the navigation map (line 5).

Then, we receive the low level information (line 7) and translate it (line 8) to the high level state. This process is also performed after each action is executed, but it is done at this point just to get the initial state. And we generate a plan with a planner (line 10).

```
ROPEM (problem, domain, planner)

  loadDomain(domain);
  loadProblem(problem);
  loadMap (problem);

  lowLevelState = receiveLowLevelState();
  highLevelState = lowToHigh(lowLevelState);

  plan = generatePlan(domain, problem, planner);

  while (!monitoringGoalsAchieved && executionCode != OK)

    while (!monitoringGoalsAchived && executionCode == OK)
      executionCode = executeAction (nextAction(plan));
      monitoringGoalsAchived = checkSampleGoals();
      lowLevelState = receiveLowLevelState();
      highLevelState = lowToHigh(lowLevelState);
    end-while

    if (executionCode == REPLAN)
      replanproblem = generateProblem(highLevelState);
      plan = generatePlan(domain, replanproblem, planner);
      executionCode = OK;
    end-if

    if (executionCode == ERROR)
      print("Unexpected Error. Execution Ended");
    end-if

  end-while

terminateExecution
```

Fig. 4. High level description of the execution algorithm.

Once we have the initial plan to be executed we enter on the *global execution loop* (lines 12-33), until the goals have been achieved, replanning is needed or there is a failure.

From line 14 to line 19 we have the *plan execution loop*, where we execute all the actions of the current plan. We execute each action (line 15) obtaining an execution code. After executing each action, goal achievement is monitored (line 16).

Once the algorithm exits the *plan execution loop* (lines 14-19), it analyzes the two exit conditions left (replanning and error) for the *global execution loop*. If there is an execution error (like a mapping mismatch or a device malfunction) it stops the execution (line 27). If the execution code indicates the need of replanning (line 21), then it generates a new PDDL problem from the current high level state (line 22), it executes the planner (line 23) with the same domain and the new problem and continues with the *global execution loop*.

Now, each module of the ROPEM architecture (Figure 5) is detailed.

A. Low Level Control

To achieve robotic-device independence, the actuators/sensors management is all done by the low level control layer that implements the basic control skills. This module receives low level action requests and sends the appropriate commands to the robot actuators, handling the corresponding communication with the control platform server. In other words, this module provides a set of basic skills that compose a low-level control server interface that is going to be used by the execution module (see Section III-B). This module is also in charge of obtaining the sensor readings from the robot after the execution of each actuator command.

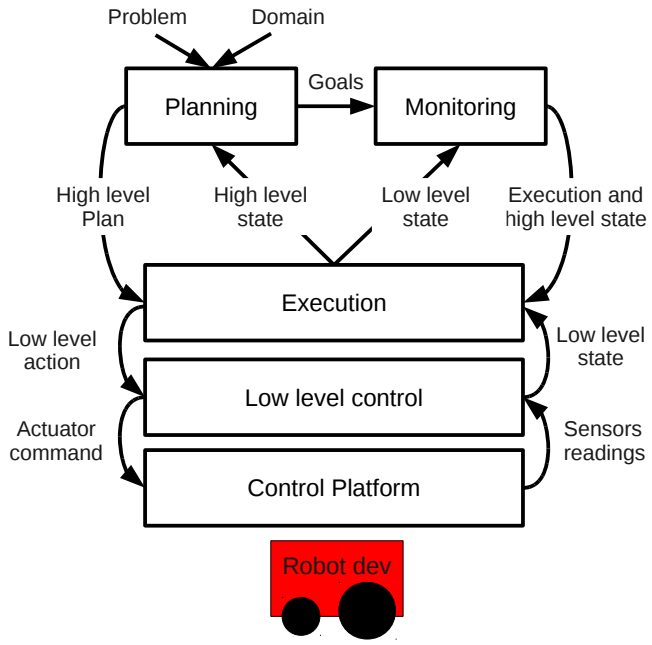

Fig. 5. ROPEM Overview.

The chosen control platform for commanding the robot is Player [11]. Player is a (TCP) network server that provides an interface for robot device (sensors/actuators) control, designed to be language and platform independent. The Player project includes simulation environments (2D, Player/Stage; and 3D, Player/Gazebo) and other useful tools such as the monitoring application *playerv*. This platform provides official support for several languages and has non-official libraries for many others, allowing ROPEM architecture to be as robot independent as possible.

Given that this module has been implemented for mobile robot bases in 2D, Player assures its reusability (with minor changes) for any planar mobile robot and has been tested with the real robot Pioneer P3DX.

B. Execution

To avoid control platform dependency, the execution is separated from the low level control. That way, our approach does not depend on control platforms or robotic devices.

The execution module receives the high level plan resulting from the execution of a deliberative planner (see Section III-D). Then, it executes one by one the high-level actions of the generated plan and receives the resulting state of the world after the execution of each action. The execution is done as follows: each high level action is decomposed into the corresponding low level actions (see Table I); then, each one of the low level actions is executed by the low level layer (see Section III-A); and once the execution of all these sub-actions is finished, the resulting low level state is translated to the corresponding high level state by the monitoring module (see Section III-C), which at the same time verifies the goal achievement.

High Level Actions	**Used Low Level Behaviours**
Navigate	`moveTowardX`, `moveTowardY`, `turnRight`, `turnLeft`
Calibrate and Sample Rock/Soil	`findBlob`, `gotoBlob`, `bumpCenter`
Communicate Data	`sendEmail`
Take Image	`saveFrame`
Drop	No low level behaviour (see section IV-C)

TABLE I
LOW LEVEL BEHAVIOURS CONFORMING EACH HIGH LEVEL ACTION.

C. Monitoring

It is difficult to predict the result of executing an action in non-deterministic planning domains such as the robot control ones, where the environment is dynamic. So, we need a module that supervises the execution and detects when it can not continue with the execution of the initial plan. Monitoring can be done at the low level or at the high level. For our work, we focus on high level supervision using a monitoring module.

This module carries out the supervision of the plan execution and the achievement of goals by receiving information from the execution module. To verify the execution state, the monitoring module receives (from the planning module, described in Section III-D) the goals of the problem to be solved. On each execution step, aside from checking whether the goals have been fulfilled, it also determines whether replanning is required.

The current re-planing policy verifies if the preconditions of the next action to be executed are satisfied. For example, in the action `navigate` (shown in Figure 6), where the rover `?x` navigates from `?y` waypoint to `?z`, it verifies the following three preconditions (`:precondition`) to determine if we need to re-plan or not: check that the rover is available (not performing another task), that it can go to the destination waypoint and that this waypoint is visible from the current one.

```
(:action navigate
 :parameters (?y - waypoint ?z - waypoint
              ?x - rover)
 :precondition (and (can_traverse ?x ?y ?z)
                    (available ?x) (at ?x ?y)
                    (visible ?y ?z))
 :effect (and (not (at ?x ?y)) (at ?x ?z)))
```

Fig. 6. PDDL `navigate` action description.

As mentioned in the previous section (Section III-B), every action (high level task) is decomposed into a set of low level behaviors. Our system supervises the sensor readings (low level information) after executing each low level action and if a failure is detected, the execution of the high level action being executed is considered erroneous. As the low level behaviours are simple and executed in a short period of time, state changes at both the low and high levels are going to be detected while executing low-level actions.

D. Planning

This module is in charge of executing the planner and generating the sequence of high level actions that are going to be executed. We use the SAYPHI planner [12]. SAYPHI is an automated planner built with the aim of integrating various machine learning techniques applied to planning. Actually SAYPHI consists of a forward search planner like Metric-FF [13], including many search algorithms. It also includes four learning sub-systems, all of them competitors in the first learning track of the International Planning Competition (IPC) [14].

Given that we are using the standard PDDL domain description from the IPC, any deliberative planner could have been used instead. The planner receives the PDDL domain and a problem specified in the same format, and returns the corresponding sequence of high level actions that have to be executed in order to achieve the goals.

The planning module is also in charge of sending the goals of the problem that is being solved to the monitoring (see section III-C) in order to supervise the execution of the proposed solution.

In case of replanning, the planning module (that already knows the domain) will receive the high level state from the execution module (see section III-B). Then the entire process will begin again using this state as the new initial state.

IV. Experiments

A first set of experiments (section IV-E) was conducted, focusing on the planning and replanning capabilities of our approach. The objective was to observe how our approach behaved with two different sources of controlled uncertainty: speed and obstacles. We executed two different problems in two maps, with different speed configurations and in presence and absence of obstacles. We wanted to test the planning/replanning capabilities and the global behaviour of the ROPEM architecture with these specific configurations. Each experiment was executed five times and the presented results are the average values obtained in all executions.

In the second set of experiments (section IV-F), a third source of simulated uncertainty was introduced to illustrate one of the issues that can be improved in the planning process. We included a temporal component in order to study the action durations, showing the importance of generating an accurate task model in AP. In particular, we added duration to the `navigate` action of the rovers domain and simulated different types of terrain affecting the real execution, illustrating the gap that can be found between the observed real world and our representation of it. Each experiment was executed once.

A. Experimental Setup

A Pioneer P3DX, equipped with sonar, bumpers and a motor-base, was used along with a Logitech Sphere cam (with PTZ capabilities). The control software was running in a PC connected via USB to the camera (usb-usb) and the robot (serial-usb).

The implemented low-level control module (see section III-A) provides an interface that allows controlling the following sensors: sonar, motor base, camera (PTZ and blob detection) and bumpers. For localization we used the odometry information provided by the motor base of the robot (x, y and yaw). This information was also used to locate the robot on the waypoint map (Section III-C).

The grid to which the waypoints were mapped was drawn on the floor, only for external monitoring during the experiments. The robot did not use this drawn grid to orient itself. Figure 7 shows the robot in the test environment.

Fig. 7. P3DX Robot (Our Rover), on the test environment (our Mars).

B. Rovers Domain Implementation

For the experiments we implemented the Rovers domain where, as stated in Section II, the objectives are to communicate a set of sample/image data to the lander, and problems involve task and path planning. For example, the PDDL formalization of a problem is detailed on Figure 8.

In line 1, the `Rover` domain is specified. Between lines 2 and 8, the objects are listed. In this problem, we use the rover (`p3dx`) and its store (`p3dxstore`), the camera (`logitechsph`), two waypoints (`wp0` and `wp1`) and one objective (`obj1`). From line 10 to 25, the initial state is described. And in lines 26 and 27 the goals (communicating an image of the only objective and the soil data of the only soil sample) are specified.

C. Domain Mapping

In order to map high-level actions and states into sensing/acting data and robot low-level behaviours, we use the following representation mapping.

Navigate (Figure 9) is mapped to turn and move behaviors, orienting the robot in the direction of the destination waypoint and moving it forward. First, we make the high level verifications (position, connection and visibility between waypoints). Then we determine the final orientation of the robot with respect to the destination waypoint and turn it into that direction, and move forward.

```
(define (problem wafexample) (:domain Rover)
  (:objects general - Lander
            colour high_res low_res - Mode
            p3dx - Rover
            p3dxstore - Store
            wp0 wp1 - Waypoint
            logitechsph - Camera
            obj1 - Objective)

  (:init (channel_free general)
         (available p3dx)
         (at p3dx wp0)
         (store_of p3dxstore p3dx)
         (empty p3dxstore)
         (equipped_for_imaging p3dx)
         (on_board logitechsph p3dx)
         (supports logitechsph high_res)
         (calibration_target logitechsph obj0)
         (visible wp0 wp1)
         (visible wp1 wp0)
         (can_traverse p3dx wp0 wp1)
         (can_traverse p3dx wp1 wp0)
         (at_lander general wp1)
         (visible_from obj1 wp1)
        (at_soil_sample wp0))
  (:goal (and (communicated_image_data obj1 high_res)
              (communicated_soil_data wp0))))
```

Fig. 8. PDDL Rovers problem description.

```
navigate (origin_wp, destination_wp)
     isAt(origin_wp)
     canTraverse(origin_wp, destination_wp)
     is_visible(origin_wp, destination_wp)
     orientateTO(destination_wp)
     moveTowards(destination_wp)
```

Fig. 9. Navigate action mapping.

Calibration and rock/soil sampling are represented by joined blob tracking and bumping actions. For instance, taking a rock sample (Figure 10) is represented as: locating a blue blob, reaching it and finally bumping that zone with the center front bumper of the robot. Specifically, blob-tracking is implemented as a simple blob-loop, with the camera panning and a blob detection proxy (provided by Player). Before executing any physical corresponding actions, the high level verifications (position and store emptiness) are performed. We used that representation for those actions, because we had no actuator (like a gripper or a robotic arm) for performing the actual actions.

```
sample_rock (sample_wp)
    isAt(sample_wp)
    isStoreEmpty()
    findblPanning(blue)
    aproachbl(blueBloob)
    sampledRock(sample_wp)
    fullStore()
```

Fig. 10. Sample-rock action mapping.

Taking an image is represented by capturing a real image with the camera at the moment of executing this action, and communication of (soil, image and rock) data is represented as sending an email. The rest of high level actions are not represented as any real physical action, because all the actuators are already in use. For example, the drop action was not included, because our robot does not have an arm that can take real samples, so there is nothing to drop.

Also, in order to use the Rovers domain, the waypoints are mapped to a grid. The mapping is done as follows: waypoints are defined as an (x, y) pair; the distance between waypoints, on the grid, is d; to differentiate between waypoints, a bounding box of d^2 is established for them; and to simplify, the bounding boxes are adjacent to each other.

More details on the high and low level states mapping is provided on the next section (Section IV-D).

D. Low/High Level States

The low level state consists of the following sensor readings: odometry information, x, y and yaw real values; bumper information, one binary value for each bumper, $b_1, \ldots, b_5$; the readings of the eight sonars, $s_1, \ldots, s_n \in \mathbb{R}$; and the information of the largest blob (x-y coordinates, top, bottom, area and color). The high level state is defined by the domain predicates: `at`, `calibrated`, `have-rock-analysis`, `have-soil-analysis`, `have-image`, etc.

The monitoring module translates the low level state to the high level state for the execution module. Robot sensor readings are translated to domain predicates after the execution of each high level action, as defined in Table II.

In the first column of Table II we mention the used low level values, in the second one we list the affected predicates and in the last one we summarize the mapping function.

The odometry information (x, y and yaw) of the low level is used to determine the current waypoint (`at` predicate) and orientation. Each waypoint has a cell on a high level grid (see Section IV-C). The current high level position is computed by checking in which cell the x, y pair is located.

To represent that a sample has been picked up and loaded in the robot store, we check that the blob of the corresponding color has been found and successfully reached (maximum size and sonar distance) and that the front center bumper has been bumped. In this case, we make true the `full` predicate for that store and the corresponding `have-xxx-analysis`. Notice that when a high level task is executed, the domain action preconditions are also checked (not mentioned on Table II).

E. Performance

We tested the ROPEM architecture performance with two different problems on the same map. In Figure 11 an example of the rock/soil samples placement and the waypoints from which the objectives are visible is shown. On that graphic representation, the gray waypoints are the external area, which is represented in order to consider the case of the rover going out of the map. Thus, rovers can be *in* that area (in case of a failure execution) but not go *into* them intentionally. The shown colors (yellow, blue and red) are the ones that were assigned to the blob-tracking behavior.

The first problem used for the experiments consists of nine goals, involving twenty waypoints (6 describing the map and 14 the exterior) and one rover (the P3DX robot). And the

Low Level	High Level	Mapping
Robot `r` in position x, y	`(at ?x - rover ?y - waypoint)`	**if** x,y in cell `w` **then** `(at r w)`
Blob, sonar and bumper data of robot `r` (with store `s`)	`(have-rock-analysis ?r - rover ?w - waypoint)` and `(full ?s - store)`. (Sample location: waypoint `w`)	**if** max blue blob reached and center bumper bumped **then** `(have-rock-analysis r w)` and `(full s)`
Blob, sonar and bumper data of robot `r` (with store `s`)	`(have-soil-analysis ?r - rover ?w - waypoint)` and `(full ?s - store)`. (Sample location: waypoint `w`)	**if** max yellow blob reached and center bumper bumped **then** `(have-soil-analysis r w)` and `(full s)`
Blob data of robot `r` (with camera `c`)	`(calibrated ?c - camera ?r - rover)`	**if** max red blob found **then** `(calibrated r c)`

TABLE II
LOW TO HIGH LEVEL STATES TRANSLATION.

second problem consists of communicating one sample of rock data, one sample of soil data and one image data.

Two kinds of tests were conducted: the *slow* experiments, that were done with a forward-speed of 0.2m/s and a turn speed of 0.2rads/s; and the *fast* experiments, that were performed with a forward-speed of 0.4m/s and a turn speed of 0.35rads/s. In the case of the experiments with obstacles, a second remotely-controlled robot was crossed in the middle of the rover trajectory five times, during the execution of different navigate actions. The moving obstacle momentarily crossed the path of the rover to simulate the case of another rover exploring a close zone, in order to test the replanning capabilities in that specific case.

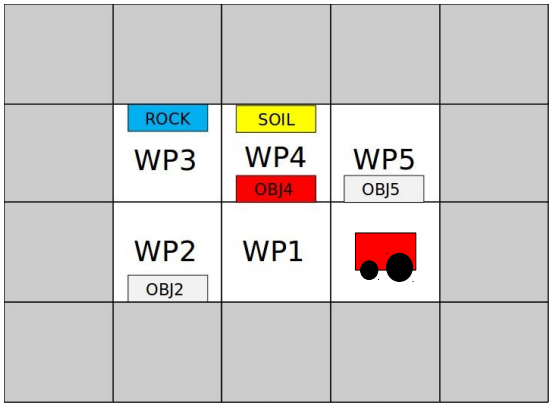

Fig. 11. Graphical representation of a problem.

1) Performance Results: Tables III and IV summarize the experimental results for the first problem. The following metrics (measure column) were established: number of initial actions, total executed actions; number of replanning steps performed; total execution time, in minutes; and total planning and replanning time, in seconds.

Without obstacles (see Table III), in both speed configurations, the total high-level executed actions is the same as in the original plan. Given that higher speeds introduce more failures in execution, in the fast executions, replanning was needed, while in the slow configuration it was not. Despite the fact that replanning was necessary because of the noise accumulation, caused by the turn and forward speedup, the execution time was still reduced.

Absence of Obstacles		
Measure	Slow	Fast
Initial actions	59±0	
Total actions	59±0	
Replanning	0±0	3±0.4
Total execution time	17±0.7m	14.7±0.5m
Planning/Replanning time	$<$ 0.5s	

TABLE III
SUMMARY OF EXPERIMENTS RESULTS FOR THE FIRST PROBLEM: ABSENCE OF OBSTACLES, WITH DIFFERENT SPEEDS.

With obstacles (see Table IV), in both speed configurations, the total high-level executed actions (71 in the slow configuration and 74 on the fast one) is significantly increased with respect to the number of initial planned actions (59). The number of replanning steps needed were the same on both configurations. Thus, the speedup did not affect significantly the total execution time. But, in the case of obstacles, we did not improve with the increase of speed.

Presence of Obstacles		
Measure	Slow	Fast
Initial actions	59±0	
Total actions	71±6.5	74±5.5
Replanning	20±9	20±5.5
Total execution time	15±0.3m	16.3±1.8m
Planning/Replanning time	$<$ 2s	

TABLE IV
SUMMARY OF EXPERIMENTS RESULTS FOR THE FIRST PROBLEM: PRESENCE OF OBSTACLES, WITH DIFFERENT SPEEDS.

In this problem, with our proposed configuration, the obstacles represented a stronger source of uncertainty, while the noise introduced by the speedup caused a smaller number of replanning episodes.

The second experiment to test the ROPEM architecture, unlike the first one, was executed only without obstacles because

the objective was to observe the behavior of the ROPEM architecture in smaller problems. Table V summarizes the experiments results for the executions of the second problem. The same metrics were used.

Speed Test		
Measure	Slow	Fast
Initial actions	26±0	
Total actions	26±0	22±2
Replanning	0±0	5±10
Total execution time	7.8±0.2m	4.8±1.5m
Planning/Replanning time	< 0.4s	

TABLE V
SUMMARY OF EXPERIMENTS RESULTS FOR THE SECOND PROBLEM. SIMPLE PROBLEM WITH DIFFERENT SPEEDS.

The solution of this problem consisted of 26 actions (initial plan), which is less than half of the previous problem solution. The difference between the total executed actions (second row) is because in some of the *fast* executions, the accumulation of noise in the odometry made the robot accidentally go into waypoints that shortened the path (i.e. moving forward two waypoints at a time). As it is a small problem, only 5 replanning steps were performed on the *fast* configuration; and no replanning was done on the *slow* one. In this problem the total execution time was reduced to half with the *fast* configuration; so the increase of speed was totally worth, taking into account the insignificant amount of time consumed by the replanning steps that the speedup noise caused.

In both problems the resulting replanning time was insignificant with respect to the tasks execution time, so there would be no need to include plan adaptation strategies at this point.

F. Navigation Time

Once the ROPEM architecture performance was tested, we decided to analyze the action durations. We improved the model by introducing a temporal component in the domain and analyzed how the task execution uncertainty and the sensor-noise caused by the environment can affect the action duration.

In order to introduce durations into the domain, for this part of the experimentation, the classical specification of the *Rovers* domain (used in the previous experiments) was modified. In Figure 12 we show the changes, with respect to the original domain, highlighted in red.

Two types of terrains (`sandy` and `rocky`) were added on lines 5 and 6. Also, a temporal component was incorporated to keep track of consumed navigation time (`navigationtime`, line 33). The time that our robot takes to traverse from one waypoint to another (with the slow speed configuration), in reality, is around 6 seconds, so the `navigationtime` fluent is increased 6 seconds every time a navigation action is executed (line 40). To our robot, that time was almost constant because we always worked in the same environment. The terrain of the hallway of our laboratory (where we performed all the experiments) does not present any irregularity. But that is not realistic. A rover navigating the surface of Mars will

```
1  (define (domain Rover)
2  (:requirements :typing)
3  (:types rover waypoint store camera mode lander objective)
4  (:predicates
5        (sandy ?x - waypoint)
6        (rocky ?x - waypoint)
7        (at ?x - rover ?y - waypoint)
8        (at_lander ?x - lander ?y - waypoint)
9        (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
10       (equipped_for_soil_analysis ?r - rover)
11       (equipped_for_rock_analysis ?r - rover)
12       (equipped_for_imaging ?r - rover)
13       (empty ?s - store)
14       (have_rock_analysis ?r - rover ?w - waypoint)
15       (have_soil_analysis ?r - rover ?w - waypoint)
16       (full ?s - store)
17       (calibrated ?c - camera ?r - rover)
18       (supports ?c - camera ?m - mode)
19       (available ?r - rover)
20       (visible ?w - waypoint ?p - waypoint)
21       (have_image ?r - rover ?o - objective ?m - mode)
22       (communicated_soil_data ?w - waypoint)
23       (communicated_rock_data ?w - waypoint)
24       (communicated_image_data ?o - objective ?m - mode)
25       (at_soil_sample ?w - waypoint)
26       (at_rock_sample ?w - waypoint)
27       (visible_from ?o - objective ?w - waypoint)
28       (store_of ?s - store ?r - rover)
29       (calibration_target ?i - camera ?o - objective)
30       (on_board ?i - camera ?r - rover)
31       (channel_free ?l - lander)
32 )
33 (:functions (navigationtime))
34
35 (:action navigate
36 :parameters (?x - rover ?y - waypoint ?z - waypoint)
37 :precondition (and (can_traverse ?x ?y ?z) (available ?x)
38                   (visible ?y ?z) (at ?x ?y))
39 :effect (and (not (at ?x ?y)) (at ?x ?z)
40             (increase (navigationtime) 6)))
41 ...
```

Fig. 12. Rovers domain with navigation time.

traverse different types of terrains and that was one of the motivations for this second set of experiments.

Based on this modified Rovers domain with navigation time, we tested the effect of the terrain types on the navigation with a simulated navigation delay. To sum up the work, it was assumed that:

- a waypoint can only be of one type of terrain,
- the delay induced by each type of terrain is constant,
- the odometry is not affected,
- and the orientation tasks do not introduce any navigation delay.

The goal was to study the effect of another controlled source of uncertainty, as was the terrain types.

As stated before, the time spent by the real robot in navigating from one waypoint to another, ignoring the previous orientation process, is of around 6 seconds. We decided to simulate how the terrain types affected this time instead of working with different terrains in reality in order to avoid introducing other sources of uncertainty. For simulating the navigation delays, `sleep` commands were used during the execution of the high level task `navigate`. Specifically, the introduced delays were the following:

- 0 seconds, for the case where the origin and destination waypoints are both sandy;
- 4 seconds, when the origin or the destination waypoint are of type rocky;

- and 8 seconds when both the origin and the destination are rocky.

The resulting total navigation times to navigate from a given waypoint to a consecutive one, are as follows: 6s for the case of sandy-sandy, 10s for rocky-sandy and sandy-rocky, and 14s for the rocky-rocky combination.

1) Navigation Evaluation: In order to study the effect of simulated terrain types, we conducted a new set of experiments in bigger maps and analyzed the navigation time of the robot.

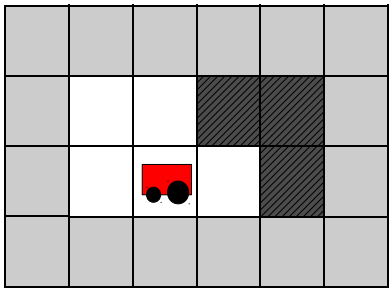

Fig. 13. Map used on the first problem for navigation experiments.

For the first experiment of this set we executed a problem on a 24 waypoints grid, where 8 waypoints represented the map where the rover is supposed to navigate and the rest (16 waypoints) represented the exterior of that map. In Figure 13 we can see the graphical representation of this map. The gray cells represent the exterior, the black cells represent the rocky waypoints and the white ones represent the sandy waypoints.

For the second and third tests we used two different problems in a new map. This time the execution was performed on a 36 waypoints map (Figure 14), with 22 waypoints representing the exterior and 14 representing the navigation map. Again, black cells are the rocky waypoints and white cells are the sandy ones.

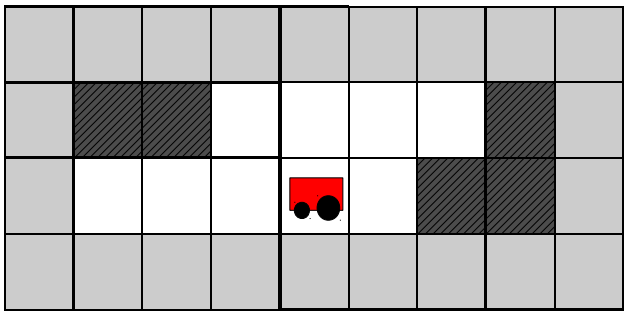

Fig. 14. Map used for the problems two and three during the navigation experiments.

We used the slow speed configuration for all the executions, and we executed one time each problem. In Table VI results are summarized; we show the initial planned actions versus the executed ones, the number of executed navigate actions and the navigation/execution time.

Navigation			
Measure	Ex1	Ex2	Ex3
Executed/Planed Actions	59/59	75/72	92/114
Navigate Actions	31	56	66
Navigate Time (Domain)	3.1m	5.6m	6.5m
Real Navigation Time	7.6m	13.9m	16.4m
Total Execution Time	8m	15m	26m
Replanning Episodes	0	2	5
Planning/Replanning Time	< 1s		

TABLE VI
RESULTS OF THE NAVIGATION EXPERIMENTS.

Execution of the first two problems (*Ex1* and *Ex2* on the Table VI) was completed, but the plan of the last problem was not successfully completed due to the accumulation of error in the odometry. With this last experiment (*Ex3* on the table) we noticed that the odometry stops being a reliable source of positioning information, to locate the robot on the map, during long runs, as expected. Thus, this first implementation approach is not scalable to big problems (as is). Although the monitoring and planning approach are useful to overcome other types of challenges, the localization is a problem that has to be solved apart.

Multiplying the `navigate` executed actions by the theoretical navigation time (6s), defined in the modified Rovers domain, we obtain the *Navigate Time (Domain)* row. And we can see that this time is far from the real navigation time in all problems (see the *Real Navigation Time* row on Table VI). This is due to the delays we introduced for terrain types. This information is not represented on the domain and this shows how the planning approach needs some help. The environment modification is not being taken into account on the planning process, so we need to include some technique that allows us to extract this knowledge during execution as, for instance, machine learning techniques that acquire domain models from execution.

V. RELATED WORK

One of the first applications of Automated Planning (AP) in robotics was for the generation and monitored execution of robot plans [15] using the classic planner STRIPS [16].

Reactive approaches have also been used for robot control, like architectures based on Reactive Actions Packages (RAPs) [17], [18]. RAP-based control systems, like other control systems, commonly suggest three levels of abstraction (execution, planning and hardware), similar to our structure. The main difference with our work is that, for these approaches, a plan is a set of RAP pre-defined tasks, so they are based on robot device-specific skills. In our approach we pull apart the low (hardware) control from the high level trying to gain platform independence. Reactive approaches usually focus on the combination of reactive behaviours, while at the moment, we focus on high level planning, using the PDDL standard combined with low level skills.

Another example of planning techniques applied to robotics, is the use of hierarchical planners [19] for robot control.

Architectures that combine hierarchical task planning with path planning, have been proposed [20]. Also hierarchical approaches have been implemented extending behavior-based architectures for robot control [21]. The disadvantage of using this type of planners is that a custom hierarchical task networks (HTN) have to be defined for each domain, complicating the domain definition.

AP techniques have also been applied to real planetary exploration, as with the Rovers that are currently on Mars [22]. This problem is particularly interesting, because it is a case in which having a planning system that provides autonomy is essential. In this case, AP techniques are not used on board the rover, but on ground. Also, they used a different planning paradigm, timeline-based planning, which lies closer to scheduling. A two-layer architecture has been used in real Rovers [4], focusing on interoperability of robot-control software, integrating a decision layer and taking into account high-level autonomy, as in our work.

T-REX [23] is an on-board system for planning and execution applied to the control of autonomous underwater vehicles in real oceanographic scientific missions. T-REX uses an specific language (NDDL) to describe the domain, and it is based on the notion of partition; planning is done at different abstraction levels by different hierarchical modules, each of one embedding a temporal constraints satisfaction-based planner. Instead, we propose to use a standard planner that reads a PDDL (Planning Domain Definition Language) domain description and is able to control the robot on-board. At the moment we are not focusing on domains where a temporal component is essential, so we can not benefit from T-REX advantages.

There have been recent developments in plan-based control of autonomous robots [24], where different approaches used plan-based high-level and Structured Reactive Controllers (SRCs), among others. Again, these approaches are closer to the hardware and therefore are tight to the robotic devices.

Control systems combining opportunistic planning and reactive techniques in the low level behaviors have been developed [25]. There have been contributions where high level actions have been defined as behaviors themselves, close to the current standard way of defining high level actions in PDDL. The advantage of our approach again is that we can benefit from any PDDL planner.

Autonomy systems for rovers control have been proposed using probabilistic planning technology [26]. Partially Observable Markov Decision Process (POMDP) approaches have been used for plan generation, taking into account action and sensing uncertainty. For our contribution we decided to focus on classical planning techniques and deal with uncertainty by using monitoring and replanning.

We are currently improving the architecture that supports our contribution in order to reach a system that fully supports planning, execution and monitoring, at low and high level, as more generic approaches that have been recently proposed [27].

VI. Conclusions

In this paper we have presented an approach for autonomous mobile robot control, that integrates automated planning (AP) techniques, execution and monitoring. We conducted a set of experiments to test the performance of our approach on a Pioneer P3DX, using the Rovers domain. We also evaluated the effect of the environment on actions durations.

Regarding the ROPEM architecture, as the chosen robotic-control platform (Player) provides support for different programming languages, our approach is language independent. Also, Player allows us to make the control code independent of the planar mobile robot bases in 2D (with minor changes). As we separated the execution from the control, we can reuse the low and high level skills.

By applying AP we benefit from the standard language PDDL that permits us to represent domains and problems, providing us with planner/domain independence taking into account action costs, state preferences, and action durations.

We are currently working on the integration of a navigation time learning module for automatic generation of navigate duration models, improving the planning process by taking into account terrain types. And including different learning modules to improve the planning process is also part of our future work.

Our medium-term goals include improving some aspects of the architecture, such as upgrading the localization system. This will allow us to test the architecture scalability. Moreover, we are going to study different sources of uncertainty to see how they affect our approach.

As part of our future work, we will try to learn probabilistic models from execution in order to apply probabilistic planning approaches.

Acknowledgments

This work has been partially supported by the Spanish MICINN under projects TIN2008-06701-C03-03, TRA-2009-008 and Comunidad de Madrid - UC3M (CCG10-UC3M/TIC-5597).

References

[1] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Amsterdam: Morgan Kaufmann, 2004.

[2] J. E. Flórez, Álvaro Torralba, J. García, C. L. López, Ángel García-Olaya, and D. Borrajo, "Timiplan: An application to solve multimodal transportation problems," in *Proceedings of "Scheduling and Planning Applications woRKshop" (SPARK). Workshop of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS'10)*, Toronto, Ontario (Canada), 2010.

[3] M. D. Rodríguez-Moreno, D. Borrajo, and D. Meziat, "An ai planning-based tool for scheduling satellite nominal operations," *AI Magazine*, vol. 25, no. 4, pp. 9–27, 2004.

[4] I. A. D. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W. S. Kim, "Claraty: An architecture for reusable robotic software," in *in SPIE Aerosense Conference*, 2003.

[5] R. R. Penner and E. S. Steinmetz, "Automated support for human mixed initiative decision and control," in *42nd IEEE Conf. on Decision and Control*, 2003, pp. 3549–3554.

[6] M. de la Asunción, L. A. Castillo, J. Fernández-Olivares, Óscar García-Pérez, A. González, and F. Palao, "Siadex: An interactive knowledge-based planner for decision support in forest fire fighting." *AI Commun.*, vol. 18, no. 4, pp. 257–268, 2005.

[7] D. Wilkins and R. V. Desimone, "Applying an ai planner to military operations planning," in *Intelligent Scheduling*. Morgan Kaufmann, 1992, pp. 685–709.

[8] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, pp. 61–124, 2003.

[9] S. Jiménez, F. Fernández, and D. Borrajo, "The pela architecture: Integrating planning and learning to improve execution." in *AAAI*, D. Fox and C. P. Gomes, Eds. AAAI Press, 2008, pp. 1294–1299.

[10] H. L. S. Younes and M. L. Littman, "Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects," *Technical Report CMU-CS-04-167*, 2004.

[11] B. Gerkey, R. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, 2003.

[12] T. De la Rosa, A. García-Olaya, and D. Borrajo, "Using cases utility for heuristic planning improvement," in *Proceedings of the 7th International Conference on Case-Based Reasoning*. Belfast, Northern Ireland, UK: Springer Verlag, August 2007, pp. 137–148.

[13] J. Hoffmann, "The metric-ff planning system: Translating "ignoring delete lists" to numerical state variables," *Journal of Artificial Intelligence Research. Special Issue on the 3er International Planning Competition*, vol. 20, 2003.

[14] T. de la Rosa, R. García-Durán, S. Jiménez, F. Fernández, A. García-Olaya, and D. Borrajo, "Three relational learning approaches for lookahead heuristic search," in *Proceedings of the Workshop on Planning and Learning of ICAPS09*, Thessaloniki (Greece), September 2009.

[15] R. Fikes, "Monitored execution of robot plans produced by strips," in *Processing 71 IFIP Congress 1971*. Stanford Res. Inst., Menlo Park, CA, USA: IFIP, 1972, pp. 189–94.

[16] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[17] R. J. Firby, "Adaptive execution in complex dynamic worlds," Ph.D. dissertation, Yale University, New Haven, CT, USA, 1989.

[18] R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack, "Experiences with an architecture for intelligent, reactive agents," *Journal of Experimental and Theoretical AI*, vol. 9, pp. 237–256, 1995.

[19] B. Morisset and M. Ghallab, "Learning how to combine sensory-motor modalities for a robust behavior," in *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents,*. London, UK: Springer-Verlag, 2002, pp. 157–178.

[20] J. Guitton, J.-L. Farges, and R. Chatila, "A planning architecture for mobile robotics," *AIP Conference Proceedings*, vol. 1019, no. 1, pp. 162–167, 2008.

[21] M. N. Nicolescu and M. J. Mataric, "A hierarchical architecture for behavior-based robots," in *In Proc., First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002, pp. 227–233.

[22] J. L. Bresina, A. K. Jónsson, P. H. Morris, and K. Rajan, "Mixed-initiative activity planning for mars rovers," in *IJCAI*, 2005, pp. 1709–1710.

[23] C. McGann, F. Py, K. Rajan, J. Ryan, and R. Henthorn, "Adaptive control for autonomous underwater vehicles," in *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*. AAAI Press, 2008, pp. 1319–1324.

[24] M. Beetz, "Towards comprehensive computational models for plan-based control of autonomous robots." in *Mechanizing Mathematical Reasoning*, ser. Lecture Notes in Computer Science, D. Hutter and W. Stephan, Eds., vol. 2605. Springer, 2005, pp. 514–527.

[25] V. Matellán and D. Borrajo, "Abc2 an agenda based multi-agent model for robots control and cooperation," *J. Intell. Robotics Syst.*, vol. 32, pp. 93–114, September 2001.

[26] T. Smith, "Rover science autonomy: probabilistic planning for science-aware exploration doctoral consortium thesis summary," in *Proceedings of the 20th national conference on Artificial intelligence - Volume 4*. AAAI Press, 2005, pp. 1660–1661.

[27] V. Alcázar, C. Guzmán, G. Milla, D. Prior, D. Borrajo, L. Castillo, and E. Onaindía, "PELEA: Planning, learning and execution architecture," in *Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'10)*, Brescia (Italia), December 2010.