

Autonomous Mobile Robot Control and Learning with the PELEA Architecture

Ezequiel Quintero

Universidad Carlos III de Madrid
equinter@inf.uc3m.es

Vidal Alcázar

Universidad Carlos III de Madrid
valcazar@inf.uc3m.es

Daniel Borrajo

Universidad Carlos III de Madrid
dborrajo@ia.uc3m.es

Juan Fdez-Olivares

Universidad de Granada
faro@decsai.ugr.es

Fernando Fernández

Universidad Carlos III de Madrid
ffernand@inf.uc3m.es

Ángel García-Olaya

Universidad Carlos III de Madrid
agolaya@inf.uc3m.es

César Guzmán

Universidad Politécnica de Valencia
cguzman@dsic.upv.es

Eva Onaindía

Universidad Politécnica de Valencia
onaindia@dsic.upv.es

David Prior

Universidad de Granada
dprior@decsai.ugr.es

Abstract

In this paper we describe the integration of a robot control platform (Player/Stage) and a real robot (Pioneer P3DX) with PELEA (Planning, Execution and LEarning Architecture). PELEA is a general-purpose planning architecture suitable for a wide range of real world applications, from robotics to emergency management. It allows planning engineers to generate planning applications since it integrates planning, execution, re-planning, monitoring and learning capabilities. We also present a relational learning approach for automatically modeling robot-action execution durations, with the purpose of improving the planning process of PELEA by refining domain definitions.

Introduction

Automated Planning (AP) has been successfully applied to different real-world problems, such as robot control (McGann et al. 2009). Some of the benefits of applying AP to robotics are: it provides long-term deliberative reasoning, there is a standard representation language (PDDL (Fox and Long 2003)), and it allows users to exploit models that were previously employed in other types of tasks. In AP it is typically assumed that a complete and perfect domain description can be defined, but in robotics this is rarely the case. As the generation of accurate robot control task descriptions for planning is usually a complex task, abstract models are used instead. Creating these models is not easy either and for this reason, other AI techniques such as Machine Learning (ML) are usually used to support models generation.

To take advantage of the benefits of AP we introduce the use of the general-purpose architecture PELEA (Alcázar et al. 2010) as an autonomous mobile robot control system. PELEA is a domain-independent platform that currently includes state-of-the-art components for performing a wide range of planning tasks. Among other features PELEA allows users to: plan using several planning paradigms (classical, temporal, hierarchical, probabilistic...), control robot task execution independently of the robot control platform and devices, monitor the correct plan execution, resolve uncertainty by re-planning when needed, and learn planning

knowledge. In this paper we want to show that this general-purpose architecture can be used in robotics and that it allows to take advantage of AP technology. To test PELEA as a robot control system we worked with the *Rovers* domain from the International Planning Competition (IPC¹). This domain is a simplified version of the planning tasks performed by the Mars Rovers.

In many domains, the actions costs depend on the context (state) where that action is applied. And the specific conditions of the state that makes an action have one cost or another may be difficult to predict when modelling the domain. To address this challenge, we present a learning approach to be included in the PELEA architecture. We focus on modelling action durations (cost equal to execution time) on unknown terrains, like the ones that could exist in other planets such as Mars. Even if we have details on how the robot performs on Earth, the actual execution of actions on Mars (or on an unknown environment) might be quite diverse. Thus, we propose the use of ML to automatically acquire that knowledge from actual actions execution. Although we focus on the Rovers domain, the studied approach could be useful in other mobile-robotics domains, including all kinds of tasks. Action durations (or similar metrics) usually depend on unpredictable factors such as weather conditions, presence of other interacting agents, navigation terrain, materials to be handled, etc.

This paper is organized as follows. First, PELEA architecture is summarized. Then, the mobile robot control integration is described. After that, a learning technique is presented. Then, we evaluate the learning proposal. And finally, related work and conclusions are discussed.

PELEA Architecture

PELEA architecture (Alcázar et al. 2010) includes components that allow to integrate planning, execution, monitoring, re-planning and learning techniques in a dynamic way. There are two main types of reasoning: high-level (mostly deliberative) and low-level (mostly reactive). This is common to most robotics applications and reflects the separation

¹IPC: <http://ipc.icaps-conference.org/>

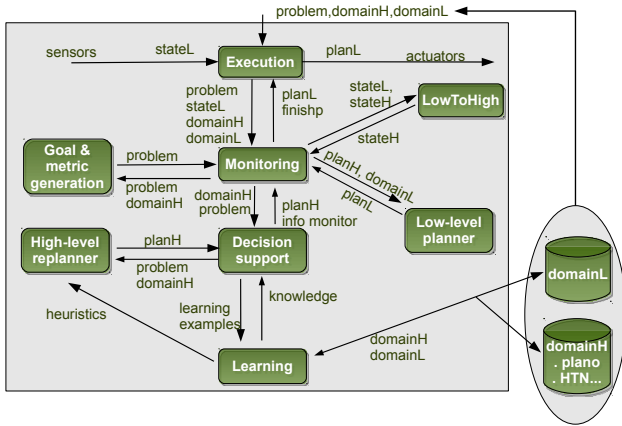


Figure 1: Architecture of PELEA.

between a reactive component and a deliberative component. In our architecture, these are simply two planning levels. As shown in Figure 1, PELEA is composed of eight modules that exchange a set of Knowledge Items (KI) during the reasoning and execution steps. The main KIs that we have used are: *stateL*, low-level state composed of the sensory information; *stateH*, abstracted high-level state obtained from *stateL*; *goals*, set of high-level goals to be achieved; *metrics*, metrics that will be used in the high-level planning process; *planH*, set of high level plans; *domainH*, definition of the model for high-level planning; *domainL*, definition of behaviors (skills) for low-level planning; *learning examples*, set of training instances used by the learning component to acquire knowledge; *heuristics*, knowledge derived from a learning process used in future planning episodes; and *monitoringInfo*, parameters that help to perform the monitoring process. The PELEA architecture is controlled by a module, called Top-level control, which coordinates the execution and interaction of the Execution and Monitoring modules. PELEA uses a two-level knowledge approach. The high-level knowledge describes general information, actions in terms of its preconditions and effects, and typically represents an abstraction of the real problem. High-level knowledge is concerned with the description of the high-level domain, problems, goals and metrics, and they are required for the purpose of planning sequences of actions, and for the modifications of these sequences (repair or re-planning). We use PDDL to represent this information. However, since high-level knowledge descriptions are rarely directly executable they must be complemented by the low-level knowledge, which specifies how the operations are actually performed in terms of continuous change, sensors and actuators. Low-level knowledge describes the more basic actions in the simulated or real world, and it is typically concerned with specific rather than general functions, and how they operate. The low-level knowledge is read from the environment through the sensors in the Execution module. The environment is either a hardware device, a software application, a software simulator or the input from an user. Now, the main

modules of the architecture will be described.

Execution Module. The starting point of PELEA is the Execution module, which is initialized by the Top-level controller, receiving a high/low-level domain, and a problem. The Execution module keeps only the static part of the initial state, given that the dynamic part, called *stateL*, will come from the environment through sensors. It is in charge of receiving the new low-level state and sending out the next actions to be executed at each step.

Monitoring Module. Both the problem (*stateL*) and domain definition are sent by the Top-level control to the Monitoring module to obtain a high-level plan (*planH*). Then, this plan is translated into a low-level plan (*planL*). The actions in *planL* are finally sent to the Execution module. The modules LowToHigh and Low-level planner are only used in case the domain is modeled at the high/low levels. Otherwise, the Monitoring calls directly the Decision Support to obtain a high-level plan (*planH*). Once the Monitoring module receives the necessary knowledge (state, problem and domain), it starts the monitoring process. The first step of the plan monitoring is to check whether the problem goals have already been achieved (*goalsL* and *goalsH* in case we are dealing with the two processes). If so, the plan execution finishes; otherwise, the Monitoring module begins with the first iteration of the plan monitoring.

Low-level planner. The Monitoring module, with the help of the Low-level planner module, generates a set of executable low-level actions (*planL*). If the Low-level planner module is not being used, the Monitoring assumes that the high-level actions in *planH* are executable, and they are directly sent to the Execution module.

LowToHigh Translator. It is in charge of translating the low-level state (*stateL*) into a high-level state (*stateH*).

Decision Support Module. It selects the variables to be observed by Monitoring and takes the decision of repairing or re-planning by an Anytime Plan-Adaptation approach (Garrido, Guzman, and Onaindía 2010). It also communicates the Monitoring module with the High-level Planner module and retrieves training instances from the execution and the plans to be sent to the Learning module.

Learning Module. It infers knowledge from a training set sent by the Decision support module. The knowledge can be used either to modify the domain planning model or to improve the planning process (heuristics).

Currently PELEA integrates the following environments: temporal probabilistic simulator, developed within the project that allows users to simulate temporal probabilistic domains in the spirit of MDPSim (Younes and Littman 2004); Virtual Robot Simulator (VRS²), which is a freeware software suite for robotics applications; Alive (Fernández et al. 2008), an open platform for developing social and emotion-oriented applications; and TIMI (Florez et al. 2010), a planning tool for real logistic problems.

Mobile Robot Control with PELEA

In this section we summarize the adjustments made in PELEA in order to support the Pioneer P3DX control. The

²VRS: <http://robotica.isa.upv.es/virtualrobot/>

three modules that required extensions were: Execution, Low Level Planner and LowToHigh.

Execution Module (EM). Actuators/sensors management is implemented in this module as a set of basic control skills. EM sends low level action requests to the robot control interface, which sends the appropriate commands to the robot actuators, handling the communication with the control platform server. After each action is executed, it reads the new *stateL* from the sensors. The chosen control platform for commanding the robot and for which we implemented a control interface to interact with the EM is Player/Stage (Gerkey, Vaughan, and Howard 2003). It is a TCP-based network server that provides an interface for robot device (sensors/actuators) control, designed to be language and platform independent.

Low-level Planner. Implements a planner translator for the *Rovers* domain and the Pioneer P3DX. Each high level domain action is decomposed into the corresponding low level actions (behaviours), executable by the robot (See table 1). In this work, high level actions correspond to the *Rovers* domain and low level skills correspond to the robot.

High Level Actions	Used Low Level Behaviours
Navigate	<code>moveTowardsXY, turnRight, turnLeft</code>
Calibrate and Rock/Soil Sampling	<code>findBlob, gotoBlob, bumpCenter</code>
Communicate Data	<code>sendEmail</code>
Take Image	<code>saveFrame</code>
Drop	No low level behaviour

Table 1: Low level skills used for the Rovers domain actions.

LowToHigh. Low/high translation (for the P3DX robot with the Rover domain) is implemented as shown in table 2. 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, \dots, b_5 ; the readings of the eight sonars, $s_1, \dots, s_n \in \mathbb{R}$; 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 odometry information (x , y and yaw) of the low level is used to determine the current waypoint (`at` predicate, on the Rovers domain) and orientation. Each waypoint has a cell on a high level grid. 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 set to true the `full` predicate for that store and the corresponding `have-xxx-analysis` literal. We used that representation for those actions, because we had no actuator (like a gripper/robotic-arm) for performing the actual actions. Notice that when a high level task is executed the domain action preconditions are also checked (not mentioned on Table 2).

Learning Action Durations

In this section, we present a learning technique, intended to be part of the Learning Module of PELEA. The approach

has been tested with a real robot using a previous architecture, similar to PELEA (Quintero et al. 2011), and remains to be integrated in the architecture. Machine Learning is commonly used to improve the planning process. One of the biggest advantages of applying it to robotic control based on AP is that we can increase the autonomy of the system and enhance its adaptation capability. The aim of this preliminary work is to improve the plan generation and execution by acquiring knowledge from the real world. We try to overcome the modeling difficulties by automatically improving the domain model, focusing on the adaptation of tasks to specific environments.

When applying AP to mobile robot control, having a precise navigate action description is crucial in any domain. Describing a model that accurately represents the navigation task is a challenge. Specially if we want to take into account characteristics as duration or cost, features that state-of-the-art planners can deal with and that are really useful for the planning process. In this work we learn the duration of the navigate action on a real P3DX robot to show that it is feasible to improve the action model by learning action durations. A rover navigating the surface of Mars will probably traverse unknown terrain types and it seems relevant to study whether these types of land may lead to different navigation times. Also, it is difficult to establish precise estimations for those times from Earth without knowing the exact characteristics of the terrain. This is just an example of knowledge that is needed in the domain model for which we do not have a priori precise estimations.

Inducing regression trees is a well-known approach to build models for numeric variables, making it particularly appropriate for learning durations. Given that AP domain models are expressed in predicate logic, Relational Learning (RL) is an appropriate technique to learn plan-action duration models through regression, as shown in previous work (Lanchas et al. 2007). The general learning process used in this work can be summarized as follows: knowledge gathering, where examples are extracted from plan executions; model learning, where the previous observations are used to induce the durations models with relational regression; and knowledge usage, where we integrate the learned knowledge with the previous domain model in PDDL.

For experimentation, the classical specification of the *Rovers* domain was modified (Changes shown in bold in Figure 2). This IPC domain provides a scenario to study a possible navigation time dependency on terrain types and to make a relevant contribution to other mobile-robotic domains. Two types of terrains (`sandy` and `rocky`) were added on line 8. Also, a temporal component was incorporated to keep track of the navigation time (`navigationtime`, line 11). The time that the robot needs to go from one waypoint to another is 6 seconds, so the `navigationtime` fluent is increased 6 seconds every time a navigation action is executed (line 18). We used this modified domain to test the effect of terrain types on the navigation with simulated delays on a real robot. To sum up the work, it was assumed that: a waypoint only belongs to 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 navi-

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 2: Low to high level states translation.

```

1 (define (domain Rover)
2 (:requirements :typing)
3 (:types rover waypoint store camera mode lander objective)
4 (:predicates
5   (at ?x - rover ?y - waypoint) (available ?r - rover)
6   (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
7   (visible ?w - waypoint ?p - waypoint)
8   (sandy ?x - waypoint) (rocky ?x - waypoint)
9   ...
10 )
11 (:functions (navigationtime))
12
13 (:action navigate
14 :parameters (?x - rover ?y - waypoint ?z - waypoint)
15 :precondition (and (can_traverse ?x ?y ?z) (available ?x)
16                  (visible ?y ?z) (at ?x ?y))
17 :effect (and (not (at ?x ?y)) (at ?x ?z)
18             (increase (navigationtime) 6)))
19 ...

```

Figure 2: Rovers domain with navigation time.

gation delay. Instead of modifying the terrain of the hallway where we performed the experiments, we used `sleep` commands during the `navigate` action execution to simulate navigation delays. 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 waypoints are rocky. For example, to navigate from a sandy waypoint to a consecutive sandy one, the expected navigation time is 6s of navigation plus 0s of *sandy-sandy* delay. The resulting total navigation times obtained by adding the terrain type delays (mentioned just above: 0s, 4s and 8s) to the actual navigation time (6s) are as follows: 6s for the case of *sandy-sandy*, 10s for *rocky-sandy* and *sandy-rocky*, and 14s for *rocky-rocky*. The `navigationtime` fluent defined allows us to compute the planned navigation time.

For learning, we use the relational learning tool TILDE (Blockeel and Raedt 1998) to build regression trees representing the extracted knowledge. The inputs of TILDE are a language bias and a knowledge base, and the output a set of prolog rules representing the inferred knowledge. In the language bias we specify the learning domain and in the knowledge base we provide the learning examples of the target concept (in our case: the duration of the `navigate` action). In our approach the language bias is manually extracted from the domain and the learning examples are obtained from planning and executing different problems with PELEA, automatically collecting observations from execution. Once relational trees are generated, new action models are created by automatically translating the TILDE resulting prolog rules into the domain action description in

PDDL. For the preliminary experiments we just focus on the navigation times, so learned rules only affect this action. TILDE configuration and rule translation is based on previous works (Quintero 2011; Lanchas et al. 2007).

As we did with our robot in-doors, in the real rovers, examples could be obtained during its exploration and the learning step could be performed on or off-board the robots. Training episodes could be obtained before sending rovers on the exploration missions and later refined during the mission.

Preliminary Learning Evaluation

A real Pioneer P3DX, equipped with sonar, bumpers and a motor-base, was used along with a Logitech Sphere cam (with PTZ capabilities). To locate the robot we use the odometry information (x, y, yaw) provided by the P3DX motor-base. And experiments were performed with a forward-speed of 0.2m/s and a turn speed of 0.2rads/s, in the hallway of our laboratory.

Total Navigation Time

For this experiment we use a single problem with two different maps. The solution for this problem is of length 21, where 11 of the 21 planned actions are of type `navigate`.

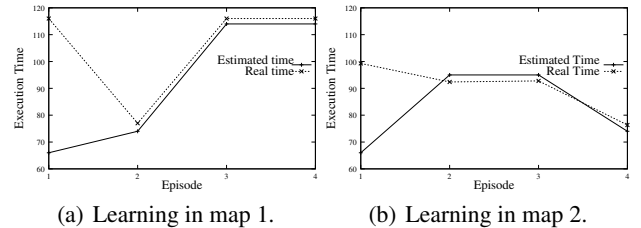


Figure 3: Estimated vs real navigation times.

In this experiment we study the convergence of the expected navigation time to the real one. The objective is to compare how close the estimated-navigation-time (with a learned domain) can get to the real-navigation-time. We performed online learning and generated a new model on each episode, that was used as the starting point for the next one. The knowledge base grew progressively with the examples of each episode. The obtained results are summarized in Figure 3. In Figure 4, the navigation maps are shown. Black cells are the rocky waypoints, white cells the sandy ones and gray cells represent the exterior of the map (for supervision).

Figure 3(a) presents the results for the navigation map with two adjacent rocky waypoints and six sandy ones (map

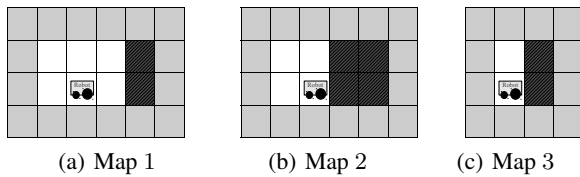


Figure 4: Navigation maps.

1: Figure 4(a)). We can observe that in the first episode the model is not accurate and the real navigation time is very different from the estimated time. In the second episode (with 22 training examples), as well as in the following episodes, the estimated time by the learned models is closer to the real times. In Figure 3(b) we can see the results for the execution of the same problem for the map of Figure 4(b). In the first episode we get a slightly worse model, although we obtain similar results in the following episodes.

Navigation Time for Terrain Combinations

For this experiment we use a navigation map of four waypoints (Figure 4(c)), where two adjacent tiles are rocky and the other two sandy. To traverse the whole map only four navigation actions are needed, and these four actions generate an example of each possible navigation case (*sandy-sandy*, *rocky-sandy*, *sandy-rocky* and *rocky-rocky*).

Ten episodes were executed, where each episode was a complete navigation of the map. Each episode generated four examples (one of each type), so in the tenth episode we had ten examples of the corresponding case. Unlike the previous experiment, in this one no learning was performed during execution. The modified Rovers domain from Fig-

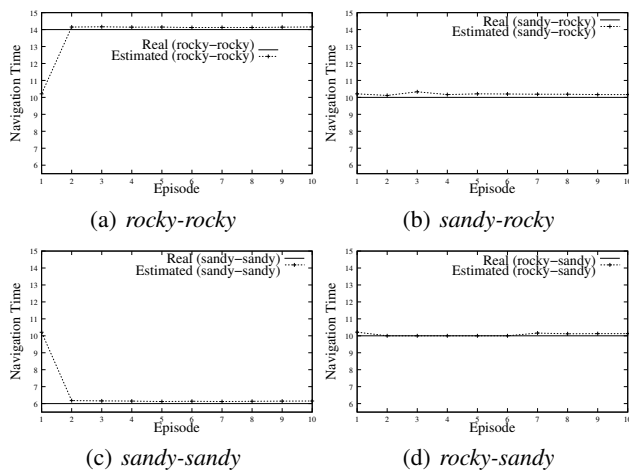


Figure 5: Learned traversing times.

ure 2 was used to analyze how the number of examples affected the learning process. In Figure 5 we show the results for this experiment. For each episode (x -axis) four additional examples are generated (4 in the first episode, 8 in the second and so on). y -axis shows the real navigation time

in continuous line and the estimated time in dotted line. In Figure 5(a) the learned time for the navigation time between two rocky waypoints is shown (*rocky-rocky*). In Figure 5(b) we can see the evolution of the learned time for the navigation from a sandy waypoint to a rocky one (*sandy-rocky*), while the *rocky-sandy* case is shown in Figure 5(d). In Figure 5(c) we observe the learned time for navigation between sandy waypoints (*sandy-sandy*). In all the graphics we can see that initially a similar duration is learned (approximately 10 seconds). The reason is that, at that point, we only have enough training data to learn the mixed terrain type navigation (*rocky-sandy* and *sandy-rocky*) time. As the robot is traversing a map with two adjacent rocky waypoint and two adjacent sandy ones, in the first episode we only have one example of each navigation case; but as the *rocky-sandy* and *sandy-rocky* navigation durations are the same, we have two examples with similar durations (close to 10s). That is the value that is learned at the beginning of the process. This causes that in the first episode of Figure 5(a) and Figure 5(c) we learn a slightly distanced value from the real navigation time, because the estimated duration for these cases (*sandy-sandy* and *rocky-rocky*) is far from 10s (is 4s and 14s). This is also the reason for the good learned times on the first episode of the other two learning cases (Figure 5(b) and Figure 5(d)). However, results are close to the real time in all the showed graphics. So, in this specific scenario, isolated from external learning noise, under controlled conditions, we can learn action durations close to the real ones with very few examples.

Related Work

There has been previous work that defines generic architectures used for different purposes. Examples can be found in space and robotics applications with platforms like Mapgen (Ai-Chang et al. 2004) and APSI (Cesta et al. 2009). These types of platforms are usually designed for particular planning techniques. The goal of the PELEA project is to build a component-based architecture able to exploit different planning techniques and perform execution, monitoring, learning in an integrated way, in the context of PDDL-based and HTN-based planning and suitable for a wide range of planning problems.

Model and Test based Transformational Learning (MTTL) has been used to improve the performance of a robot control system by autonomously adapting it to specific tasks. XFRMLEARN (Beetz and Belker 2000) integrates MTTL into a controller using reactive parametrization (SRPAs). This implementation has successfully improved the path planning process on an indoor navigation task. Our work has also been tested indoor, but without a specific path planning component. In the XFRMLEARN approach, specific parameters (such as velocity and position) are learned to improve a reactive controller, while in our contribution we learn action durations to refine a high level deliberative planning process. Behavior Models for robot execution control have successfully been learned with structured stochastic processes as the Dynamic Bayesian Network formalism. It has been shown that this technique can be used to learn models for behaviors with controllable parameters, such as reactive navigation (Infantes, Ingrand, and Ghallab 2006).

Even though stochastic methods allow robots take into account environment uncertainty during the learning process, we worked with deterministic techniques because we are currently using deterministic planning and dealing with the non-determinism during execution by monitoring and re-planning. Classical versions of both regression and decision trees have been used for action modelling in autonomous robots in ROGUE (Haigh and Veloso 1999). ROGUE used decision trees to acquire rules that prioritize its activities according to the values of its sensors. It learned rules that are compiled into heuristics (in the form of control rules) used when planning, while in our work we change the domain model. ROGUE used feature-based ML and we use relational ML instead. Notice that PELEA will be able to integrate both learning approaches.

Conclusions

In this paper we have introduced the generic AP-based architecture PELEA as a robot control system. PELEA integrates planning related processes, such as sensing, planning, execution, monitoring, re-planning and learning. We successfully tested it as a robot control system with a real robot (Pioneer 3DX) in scenarios under controlled conditions. As part of this work, we also presented a learning technique to improve the robot control and showed its performance on a specific robotic domain. We presented preliminary experiments that show that is feasible to perform plan generation/execution with knowledge acquisition from the environment, focusing on the adaptation of the navigation task. Specifically, we learned the navigate action duration on a P3DX robot depending on terrain types from a modified PDDL domain.

Acknowledgments

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

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B.; Dias, W.; and Maldague, P. 2004. MAPGEN: Mixed-initiative planning and scheduling for the Mars Exploration Rover mission. *IEEE Intelligent Systems* 19(1):8–12.

Alcázar, V.; Guzmán, C.; Prior, D.; Borrajo, D.; Castillo, L.; and Onaindia, E. 2010. PELEA: Planning, learning and execution architecture. In *Procs. of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'10)*.

Beetz, M., and Belker, T. 2000. Autonomous environment and task adaptation for robotic agents. In Horn, W., ed., *Procs. of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, 648–652.

Blockeel, H., and Raedt, L. D. 1998. Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101(1-2):285–297.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2009. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *Procs. of the 21st Innovative Applications of Artificial Intelligence Conference, Pasadena*.

Fernández, S.; Asensio, J.; Jiménez, M.; and Borrajo, D. 2008. A social and emotional model for obtaining believable emergent behavior. In Traverso, P., and Pistore, M., eds., *Artificial Intelligence: Methodology, Systems, and Applications*, volume 5253/2008 of *Lecture Notes in Computer Science*, 395–399. Varna, Bulgaria: Springer Verlag.

Florez, J. E.; García, J.; Torralba, A.; Linares, C.; Garcia-Olaya, A.; and Borrajo, D. 2010. Timiplan: An application to solve multimodal transportation problems. In Steve Chien, G. C., and Yorke-Smith, N., eds., *Procs. of the 2010 Scheduling and Planning Applications workshop (SPARK'10)*, 36–42.

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

Garrido, A.; Guzman, C.; and Onaindia, E. 2010. Anytime plan-adaptation for continuous planning. In *Procs. of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'10)*.

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

Haigh, K. Z., and Veloso, M. M. 1999. Learning situation-dependent rules. In *AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*.

Infantes, G.; Ingrand, F.; and Ghallab, M. 2006. Learning behaviors models for robot execution control. In *ECAI*, 678–682.

Lanchas, J.; Jiménez, S.; Fernández, F.; and Borrajo, D. 2007. Learning action durations from executions. In *Working notes of the ICAPS'07 Workshop on AI Planning and Learning*.

McGann, C.; Py, F.; Rajan, K.; and Olaya, A. G. 2009. Integrated planning and execution for robotic exploration. In *Procs. of International Workshop on Hybrid Control of Autonomous Systems*.

Quintero, E.; García-Olaya, A.; Borrajo, D.; and Fernández, F. 2011. Control of autonomous mobile robots with automated planning. *Journal of Physical Agents*.

Quintero, E. 2011. Improving plan execution on mobile robots by learning action durations. In *ICAPS 2011 Doctoral Consortium*.

Younes, H. L. S., and Littman, M. L. 2004. PPDDL1.0: An extension to pddl for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, School Of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.