In Proceedings of 96 AAAI Workshop: Intelligent Adaptive Agents

Using Perception Information for Robot Planning and Execution

Karen Zita Haigh
khaigh@cs.cmu.edu
http://www.cs.cmu.edu/~khaigh

Manuela M. Veloso
veloso@cs.cmu.edu
http://www.cs.cmu.edu/~mmv

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present ROGUE, an integrated planning and executing robotic agent. ROGUE is designed to be a roving office gopher, doing tasks such as picking up & delivering mail and returning & picking up library books, in a setup where users can post tasks for the robot to do. We have been working towards the goal of building ROGUE as a completely autonomous agent which can learn from its experiences improving its own behaviour. In this paper, we focus on describing ROGUE's capabilities in executing and processing perception information, including: (1) the generation and execution of a plan which requires observation to make informed planning decisions, and (2) the monitoring of execution for informed replanning. ROGUE is implemented and functional on a real indoor robot.

Introduction

We have been working towards the goal of building an autonomous robot that is capable of planning and executing high-level tasks in a dynamic environment. To achieve this end, we have been building an integrated framework, ROGUE, which combines PRODIGY, a planning and learning system (Veloso *et al.* 1995), with Xavier, an autonomous indoor robot (O'Sullivan & Haigh 1994). We aim to create a complete autonomous agent capable of planning, executing and learning in a dynamic real world environment.

One of the goals of the Xavier project is to have the robot move autonomously in an office building reliably performing office tasks such as picking up and delivering mail and computer printouts, returning and picking up library books, and recycling cans (Simmons 1994a).

Our on-going contribution to this ultimate goal is at the high-level reasoning of the process, allowing the robot to efficiently handle multiple interacting goals, and to learn from its experience. ROGUE receives tasks from users, asks PRODIGY to generate a plan for the goals, commands Xavier to execute the plan, monitors the plan's execution, and replans for any perceived failures.

Other researchers investigate the problem of interleaving planning and execution (including (Agre & Chapman 1987; Firby 1994; Hammond, Converse, & Martin 1990; McDermott 1978)). We build upon this work and pursue our investigation from three particular angles:

- that of real execution in an autonomous robot;
- that of challenging the robot with multiple asynchronous user-defined interacting tasks; and
- that of using experience as a source of learning to improve the overall performance of the autonomous agent.

The learning algorithm of the system is the focus of our current work and will be the topic of future papers. Our work focusses on the interleaving of planning and execution by a real robot within a framework with the following sources of incomplete information:

- the tasks requested by the users are not completely specified,
- the set of all the goals to be achieved is not known *a priori*,
- the domain knowledge is incompletely or incorrectly specified, and
- the execution steps sent to the robot may not be achieved as predicted.

Currently, ROGUE's main features are (1) the ability to receive and reason about multiple asynchronous goals, suspending and interrupting actions when necessary (described in a previous paper (Haigh & Veloso 1996)), (2) the ability to deliberately sense the world to update its domain model, and (3) the ability to sense, reason about, and correct simple execution failures.

In this paper, we focus on presenting how ROGUE uses observation to make informed planning decisions when the domain knowledge is incomplete, and also how the system monitors execution to adapt to a changing environment, in particular when actions fail or have unexpected effects.

The paper is organized as follows: In Section 2 we introduce the ROGUE architecture, our developed integrated system. We illustrate how ROGUE uses observation of the environment to create correct plans in Section 3. We describe how ROGUE adapts to a dynamic world in Section 4, discussing failed actions as well as unexpected outcomes. We present a brief review of related work in Section 5. Finally we provide a summary of ROGUE's current capabilities in Section 6 along with a description of our future work to incorporate learning methods into the system.

General Architecture

ROGUE[1] is the system built on top of PRODIGY4.0 to communicate with and to control the high-level task planning in Xavier[2]. The system allows users to post tasks for which the planner generates a plan, delivers it to the robot, and then monitors its execution. ROGUE is intended to be a roving office gofer unit, and will deal with tasks such as delivering mail, picking up printouts and returning library books.

Xavier is a mobile robot being developed at CMU (O'Sullivan & Haigh 1994) (see Figure 1(a)). It is built on an RWI B24 base and includes bump sensors, a laser range finder, sonars and a color camera. Control, perception and navigation planning are carried out on two on-board Intel 80486-based machines. Xavier can communicate with humans via an on-board lap-top computer or via a natural language interface.

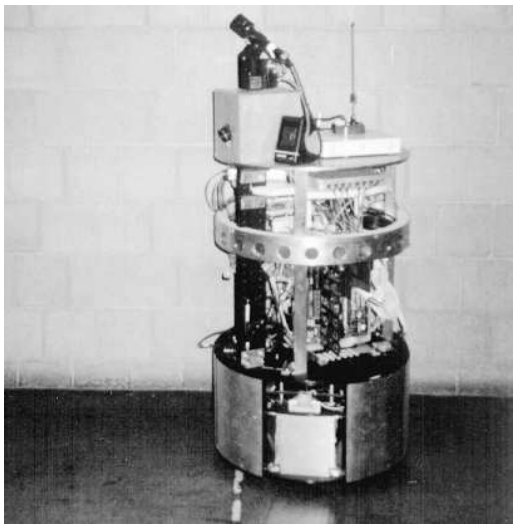
The software controlling Xavier includes both reactive and deliberative behaviours, integrated using the Task Control Architecture (TCA) (Simmons 1994b; Simmons, Lin, & Fedor 1990). TCA provides facilities for scheduling and synchronizing tasks, resource allocation, environment monitoring and exception handling. The reactive behaviours enable the robot to handle real-time local navigation, obstacle avoidance, and emergency situations (such as detecting a bump). The deliberative behaviours include vision interpretation, maintenance of occupancy grids & topological maps, and path planning & global navigation (an A* algorithm).

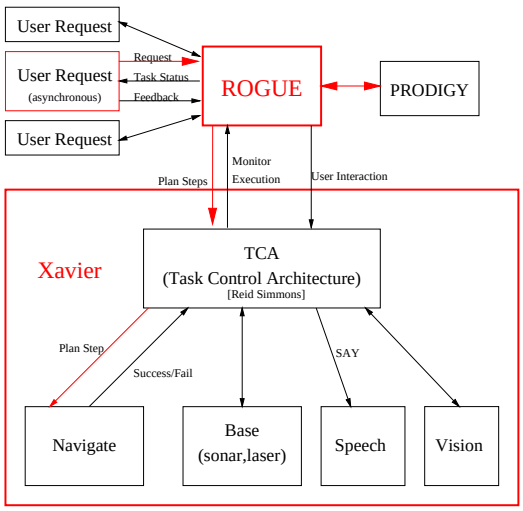
[1] In keeping with the Xavier theme, ROGUE is named after the "X-men" comic-book character who absorbs powers and experience from those around her. The connotation of a wandering beggar or vagrant is also appropriate.

[2] We will use the term Xavier when referring to features specific to the robot, PRODIGY to refer to features specific to the planner, and ROGUE to refer to features only seen in the combination.

PRODIGY is a domain-independent problem solver that serves as a testbed for machine learning research (Carbonell, Knoblock, & Minton 1990; Veloso *et al.* 1995). PRODIGY4.0 is a nonlinear planner that uses means-ends analysis and backward chaining to reason about multiple goals and multiple alternative operators to achieve the goals. The planning reasoning cycle involves several decision points, including which goal to select from the set of pending goals, and which applicable action to execute. Dynamic goal selection from the set of pending goals enables the planner to interleave plans, exploiting common subgoals and addressing issues of resource contention.

PRODIGY and Xavier are linked together using TCA as shown in Figure 1(b).





(a) (b)

Figure 1: (a) Xavier the Robot; (b) Rogue Architecture

PRODIGY maintains an internal model of the world in which it simulates the effects of selected applicable operators. Applying an operator gives the planner additional information (such as consumption of resources) that might not be accurately predictable from the domain model. PRODIGY also supports real-world execution of its applicable operators when it is desirable to know the actual outcome of an action (Stone & Veloso 1995); for example, when actions have probabilistic outcomes, or the domain model is incomplete and it is necessary to acquire additional knowledge. Some examples of the use of this feature include shortening combined planning and execution time, acquiring necessary domain knowledge in order to continue planning (*e.g.* sensing the world), and executing an action in order to know its outcome and handle any failures. ROGUE uses the execution feature of PRODIGY to send commands to the robot, monitoring the outcome of the actions, and updating PRODIGY's domain knowledge as necessary.

Model Updates Through Observation

In this section we present ROGUE's behaviour in more detail. We describe in particular the interaction between the planner and the robot, showing how symbolic action descriptions are turned into robot commands, as well as how deliberate observation is used by the system to make intelligent planning decisions.

The key to this communication model is based on a pre-defined language and model translation between PRODIGY and Xavier. PRODIGY relies on a state description of the world to plan. ROGUE is capable of converting Xavier's actual perception information into PRODIGY's state representation, and ROGUE's monitoring algorithm determines which information is relevant for planning and replanning. Similarly ROGUE is capable of translating plan steps into Xavier's actions commands.

Therefore, the fact that ROGUE needs to link a symbolic planner with a robotic executor entails the following modular capabilities:

- PRODIGY's plan steps are mapped into Xavier's commands;
- PRODIGY's state representation is a model of the execution environment;
- Xavier's perception information is abstracted into PRODIGY's state information;
- PRODIGY is capable of generating partial plans for execution in a continuous way;
- PRODIGY re-evaluates the goals to be achieved continuously based on its state information.

We illustrate the incorporation of perception information from execution into planning through an example corresponding to event 1 from the upcoming AAAI 1996 robot competition. The environment consists of three conference rooms and several offices. In this particular event, the director wishes to schedule a meeting in a conference room. The robot needs to find an empty conference room and then inform all the meeting attendees. This task is presented as a single request to ROGUE. It requires the system to incorporate observation knowledge into its planning in order to accurately and efficiently complete the task.

When ROGUE receives the task request, it spawns a PRODIGY run, giving PRODIGY relevant task information such as who are the attendees and which rooms are potential conference rooms. PRODIGY incorporates as part of its state knowledge, a topological map of the environment with the location of the rooms. It does not have complete information of the exact location of the doors.

PRODIGY starts creating a plan by alternating considering the goals and their subgoals and the different ways of achieving them. When PRODIGY finds that there are several possible conference rooms, it applies control knowledge to select the closest room to its current location, as the first one to check for availability. When PRODIGY reaches its first opportunity to simulate the effects of the operator, ROGUE interrupts and sends the action to Xavier for *real-world* execution. Each of the symbolic actions described in the domain model is mapped to a command sequence suitable for Xavier. The command sequences may be executed directly by the ROGUE module (*e.g.* an action like `finger`), or sent via the TCA interface to the Xavier module designed to handle the command. For example, the action <`GOTO-ROOM ROOM`>) is mapped to the commands (1) find the coordinates of the room, and (2) navigate to those coordinates.

Figure 2 shows a partial trace of a run. When PRODIGY applies the <`GOTO-ROOM`> operator in its internal world model (see node `n14`), ROGUE sends the command to Xavier for execution. Each line marked "`SENDING COMMAND`" indicates a direct command sent through the TCA interface to one of Xavier's modules.

The TCA command `navigateToG`(goal) (used after node `n14`) creates a path from the current location to the requested location, and then uses probabilistic reasoning to navigate to the requested goal. The module performs reasonably well given incomplete or incorrect metric information about the environment and in the presence of noisy effectors and sensors.

This example shows the use of two more TCA commands, namely `C_observe` and `C_say` (after nodes `n14` and `n18`). The first command is a direct perception action. The observation routine can vary depending on the kind of information needed. It can range from an actual interpretation

```
    n2 (done)
    n4 <*finish*>
      n5 (mtg-scheduled)
Firing prefer bindings LOOK-AT-CLOSEST-CONF-ROOM-FIRST #<5309> over #<5311>
      n7 <schedule-meeting 5309> [1]
        n8 (conference-room 5309)
        n10 <select-conference-room 5309>
          n11 (at-room 5309)
          n13 <goto-room 5309>
          n14 <GOTO-ROOM 5309>

SENDING COMMAND (tcaExecuteCommand "C_say" "Going to room 5309")
ANNOUNCING: Going to room 5309
SENDING COMMAND (TCAEXPANDGOAL "navigateToG" #(TASK-CONTROL::MAPLOCDATA 567.0d0 3483.0d0))
   ...waiting...
Action NAVIGATE-TO-GOAL finished (SUCCESS).

          n15 (room-empty 5309)
          n17 <observe-conference-room 5309>
          n18 <OBSERVE-CONFERENCE-ROOM 5309>

SENDING COMMAND (tcaExecuteCommand "C_observe" "5309")
DOING OBSERVE: Room 5309 conf-room
   ...waiting...
Action OBSERVE finished (OCCUPIED).

SENDING COMMAND (tcaExecuteCommand "C_say" "This room is occupied")
ANNOUNCING: This room is occupied

  6   n6 schedule-meeting
  7   n15 <schedule-meeting r-5311>
```

Figure 2: Trace of Rogue interaction.

of some of Xavier's sensors or its visual images, to specific input by a user. The command `C_say` sends the string to the speech board.

In this example, once the navigate module has successfully completed, ROGUE tells Xavier's vision module to observe the room. In this example, the conference room is occupied, and ROGUE updates PRODIGY's domain model. In this case, the observation sets the state information that the room is occupied. Because there are no operators in the domain model that can be used to empty a room, replanning in this case forces PRODIGY to use another conference room for the meeting (*i.e.* PRODIGY backtracks to node 7 and selects a different conference room (node 15)).

The run proceeds until ROGUE finds a conference room that is empty or until it exhausts all the available conference rooms. After a conference room is found, ROGUE proceeds to announce the attendees of the location of their meeting. The announcement is made by navigation to each individual room. The final plan executed by Xavier is shown in Figure 3. Xavier stops at all the conference rooms until it correctly identifies an empty one, and then tells all the attendees when and where the meeting will be (within 3.5 minutes in 5311). This behaviour was developed in

Xavier's simulator and then applied successfully on the real robot (and will be demonstrated at the robot competition).

```
<goto-room 5309>
<observe-meeting-room 5309>
<goto-room 5311>
<observe-meeting-room 5311>
<select-meeting-room 5311>
<goto-room 5307>
<inform-person-of-meeting director 3.5 5311>
<goto-room 5303>
<inform-person-of-meeting professor-G 3.5 5311>
<goto-room 5301>
<inform-person-of-meeting professor-S 3.5 5311>
```

Figure 3: Executed plan

Observing the real world allows the system to adapt to its environment and to make intelligent and relevant planning decisions. Observation allows the planner to update and correct its domain model when it notice changes in the environment. For example, it can notice limited resources (*e.g. battery*), notice external events (*e.g. doors opening/closing*), or prune alternative outcomes of an operator. In these ways, observation can create opportunities for the planner and it can also reduce the planning effort by pruning possibilities. Real-world observation creates a more robust planner that is sensitive to its environment.

Model Updates Through Execution Failure

The capabilities described in the preceding section are sufficient to create and execute a simple plan in a world where all dynamism is predictable. The real world, however, needs a more flexible system that can monitor its own execution and compensate for problems and failures. Any action that is executed by any agent is not guaranteed to succeed in the real world. There are several approaches to reason about the real world uncertainty. In one extreme, planners can completely ignore that the world is uncertain. They follow a deterministic model and generate a single executable plan. When execution failures occur, replanning is invoked. This is the current running mode of ROGUE. Conditional planning in the other extreme, aims at considering in the domain model all the possible contingencies of the world and plan ahead for each individual one. Because it is impossible to enumerate all the world's possible events, complete conditional planning is infeasible. Probabilistic planning falls in the middle of these two frameworks (Blythe 1994). At planning time, it accounts for the most probable contingencies and relies on replanning if unpredictable or rare events may take place and disrupt the plan execution. Probabilistic planners may increase the probability of a plan succeeding, but the domain model underlying the plan is bound to be incompletely or incorrectly specified. Not only is the world more complex than a model, but it is also constantly changing in ways that cannot be predicted. Therefore every agent executing in the real world must have the ability to monitor the execution of its actions, detect when the actions fail, and compensate for these problems.

Gil (Gil 1992), Wang (Wang 1995) and desJardins (desJardins 1994) learn or improve action models. We take the approach instead of modifying the state description to reflect the outcome

of the action, thereby forcing ROGUE to find an alternate means to achieve the goal. Ideally, an approach combining the two methods would be most appropriate for a complete autonomous agent.

ROGUE currently monitors the outcome of the `navigateToG` command. `navigateToG` may fail under several conditions, including detecting a bump, detecting corridor or door blockage, and detecting lack of forward progress. The module is able to compensate for certain problems, such as obstacles and missing landmarks, and will report success in these situations.

Since the navigate module may get confused and report a success even in a failure situation, ROGUE always verifies the location with a secondary test (vision or human interaction). If ROGUE detects that in fact the robot is not at the correct goal location, PRODIGY's domain knowledge is updated to reflect the actual position, rather than the expected position.

This update has the direct effect of indicating to PRODIGY that the execution of an action failed, and it will backtrack to find a different action which can achieve the goal. PRODIGY exhibits this replanning behaviour as an inherent part of its design: the *actual* outcome of an action must be the same as the *expected* outcome. When this expectation is invalidated, PRODIGY will attempt to find another solution.

More concretely, PRODIGY's algorithm is a state-spaced means-ends planner, which means that when it selects a plan action, it simulates the effects of the action in the domain model. PRODIGY cannot simulate the effect of an action until its preconditions are true; when they are false, PRODIGY subgoals and looks for a sequence of actions that will achieve them.

In Figure 4, for example, according to its domain model, the robot cannot deliver a particular item unless it (a) has the item in question, and (b) is in the correct location.

```
(operator DELIVER-ITEM
  (preconds
   (and (has-item <item>)
        (deliver-loc <location> <item>)
        (robot-in-room <location>)))
  (effects
   ((del (has-item <item>))
    (add (item-delivered <item>)))))
```

Figure 4: Item Delivery Operator.

Since ROGUE actually forces the real-world execution of the action, and then updates the domain model with the actual outcome of the action, PRODIGY can detect when an action failed or had an unexpected outcome. If the action failed, then the goal of the action is not deleted from the set of pending goals, and PRODIGY is forced to find an alternate means of achieving it.

In a similar manner, PRODIGY is able to detect when an action is no longer necessary. If an action unexpectedly achieves some other necessary part of the plan, then that goal is deleted from the set of pending goals, and PRODIGY will no longer plan for it, and in particular will not select any actions that will redundantly achieve it.

Finally, when an action accidentally *dis*achieves the effect of a previous action (and the change is detectable), ROGUE adds the change to PRODIGY's pending goals, forcing PRODIGY to replan.

Take for example, a coffee delivery scenario. The system picks up the coffee, adding the literal `(has-item coffee)` to its knowledge base and deleting the goal `(pickup-item coffee roomA)`.

If ROGUE is now interrupted with a more important task, it suspends the coffee delivery and does the other task. While doing the new task, the coffee gets cold, making the literal `(has-item coffee)` untrue. When PRODIGY returns to the original task, it examines the next foreseeable action: `(deliver-item coffee roomB)`, discovers that a precondition is missing (it doesn't have the coffee) and will subgoal on re-achieving it.

In this manner, ROGUE is able to detect simple execution failures and compensate for them. Our research plan includes finding methods of more informed replanning for a wider variety of more complex execution failures. The interleaving of planning and execution reduces the need for replanning during the execution phase and increases the likelihood of overall plan success. It allows the system to adapt to a changing environment where failures can occur.

Related Work

Following is a brief description of some of the robot architectures most similar to ROGUE, pointing out some of the major differences.

PARETO (Pryor 1994), can plan to acquire information and recognize opportunities in the environment (as can ROGUE), but relies on powerful, perfect sensing in a simulated world. It is also not clear how PARETO handles action failure. Plans for PARETO are created up-front by a contingency planner which needs to reason about all possible failures, whereas ROGUE exploits interleaved planning and execution and uses the actual outcome of an action to decide which branch of the plan to expand, not only reducing the total planning effort, but also the modelling effort since not all failures need to be predicted.

Gervasio's *completable planning* paradigm (Gervasio & DeJong 1996) also creates a complete plan up-front, but instead of creating contingency plans, invokes replanning upon failure. ROGUE on the other hand interleaves planning with execution reducing both the initial planning and any replanning efforts: a complete plan does not need to be constructed before execution can begin. Therefore the planner does not waste effort creating long plans that may be infeasible in a dynamic environment. In addition, ROGUE is implemented on a real robot rather than a simulated one.

Theo (Mitchell *et al.* 1990) is a system implemented on an indoor mobile robot which applies explanation-based learning to create stimulus-response rules. It does not support action interruption or uncertainty in the domain, nor does it maintain a state history or modify the action model, nor can it handle sensor noise.

RoboSoar (Laird *et al.* 1991; Tambe, Schwamb, & Rosenbloom 1995) was a system originally built on a PUMA robot arm for blocksworld problems and has since been developed into a simulated fighter pilot. The system relies on perfect sensors and a human to tell it what features of the environment are relevant for replanning.

ATLANTIS (Gat 1992) and RAP (Firby 1994), like TCA, are architectures that enable a library of behaviours and reactions to be controlled by a deliberative system. The have been used as the underlying control mechanism on a variety of robots, from indoor mobile robots (Gat 1992) to to space station repair robots (Bonasso & Kortenkamp 1996) to unmanned spacecraft (Gat 1996). We believe that ROGUE is the only such system that can support asynchronous goals, but since each of these architectures is inherently extensible, the behaviours demonstrated by ROGUE under TCA could be easily transferred to one of the other architectures.

Summary

In this paper we have presented one aspect of ROGUE, an integrated planning and execution robot architecture. In particular, we have presented the methods by which ROGUE incorporates execution information into its internal domain model to facilitate more intelligent and adaptive planning. The complete planning & execution cycle for a given task can be summarized as follows:

1. ROGUE requests a plan from PRODIGY.
2. PRODIGY passes executable steps to ROGUE.
3. ROGUE translates and sends the planning steps to Xavier.
4. ROGUE monitors execution and through observation identifies goal status; failure means that PRODIGY's domain model is modified and PRODIGY may backtrack or replan for decisions

ROGUE represents a successful integration of a classical AI planner with a real mobile robot. Currently ROGUE's features include the ability to:

- receive asynchronous goal requests from multiple users
- prioritize goals and focus planning on high priority goals until they are achieved, and then later resume work on lower priority goals;
- recognize similar goals and opportunistically achieve them;
- interleave planning & execution to acquire data and monitor execution; and
- deal with simple plan failures, such as arriving at an incorrect location.

An autonomous agent with all of these features has clear advantages over more limited agents. Although there are a small number of other integrated architectures which support some of these features, none appear to support them all.

This work is the basis for machine learning research with the goal of creating a complete agent that can reliably perform tasks that it is given. We intend to implement more autonomous detection of action failures and learning techniques to correct those failures. In particular, we would like to learn contingency plans for different situations and when to apply which correction behaviour. We also intend to implement learning behaviour to notice patterns in the environment; for example, how long a particular action takes to execute, when to avoid particular locations (*e.g.* crowded hallways), and when sensors tend to fail. We would like, for example, to be able to say *"At noon I avoid the lounge"*, or *"My sonars always miss this door...next time I'll use vision"*, or even something as apparently simple as *"I can't do that task given what else I have to do."*

PRODIGY has been successfully used as a test-bed for machine learning research many times (*e.g.* (Pérez 1995; Wang 1995; Veloso 1994)), and this is the primary reason why we selected it as the deliberative portion of ROGUE. Xavier's TCA architecture supports incremental behaviours and therefore will be a natural mechanism for supporting these learning behaviours.

References

Agre, P. E., and Chapman, D. 1987. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, 268–272. San Mateo, CA: Morgan Kaufmann.

Blythe, J. 1994. Planning with external events. In de Mantaras, R. L., and Poole, D., eds., *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 94–101. Seattle, WA: Morgan Kaufmann.

Bonasso, R. P., and Kortenkamp, D. 1996. Using a layered control architecture to alleviate planning with incomplete information. In *Proceedings of the AAAI Spring Symposium "Planning with Incomplete Information for Robot Problems"*, 1–4. Stanford, CA: AAAI Press.

Carbonell, J. G.; Knoblock, C. A.; and Minton, S. 1990. PRODIGY: An integrated architecture for planning and learning. In VanLehn, K., ed., *Architectures for Intelligence*. Hillsdale, NJ: Erlbaum. Also Available as Technical Report CMU-CS-89-189.

desJardins, M. 1994. Knowledge development methods for planning systems. In *AAAI-94 Fall Symposium Series: Planning and Learning: On to Real Applications.*

Firby, R. J. 1994. Task networks for controlling continuous processes. In *Proceedings of AIPS-94*, 49–54.

Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of AAAI-92*, 809–815.

Gat, E. 1996. News from the trenches: An overview of unmanned spacecraft for AI researchers. In *Proceedings of the AAAI Spring Symposium "Planning with Incomplete Information for Robot Problems"*, 5–12. Stanford, CA: AAAI Press.

Gervasio, M. T., and DeJong, G. F. 1996. Completable planning: A curative learning approach to the imperfect domain theory. In *Proceedings of the AAAI Spring Symposium "Planning with Incomplete Information for Robot Problems"*, 13–16. AAAI Press.

Gil, Y. 1992. *Acquiring domain knowledge for planning by experimentation.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. Also available as Technical Report CMU-CS-92-175.

Haigh, K. Z., and Veloso, M. 1996. Interleaving planning and robot execution for asynchronous user requests. In *Proceedings of the AAAI Spring Symposium "Planning with Incomplete Information"*, 35–44. AAAI Press.

Hammond, K.; Converse, T.; and Martin, C. 1990. Integrating planning and acting in a case-based framework. In *Proceedings of AAAI-90*, 292–297. San Mateo, CA: Morgan Kaufmann.

Laird, J. E.; Yager, E. S.; Hucka, M.; and Tuck, C. M. 1991. Robo-Soar: An integration of external interaction, planning, and learning using Soar. *Robotics and Autonomous Systems* 8(1-2):113–29.

McDermott, D. 1978. Planning and acting. *Cognitive Science* 2.

Mitchell, T. M.; Allen, J.; Chalasani, P.; Cheng, J.; Etzioni, O.; Ringuette, M.; and Schlimmer, J. 1990. Theo: A framework for self-improving systems. In VanLehn, K., ed., *Architectures for Intelligence.* Hilllsdale, NJ: Erlbaum.

O'Sullivan, J., and Haigh, K. Z. 1994. *Xavier.* Carnegie Mellon University, Pittsburgh, PA. Manual, Version 0.2, unpublished internal report.

Pérez, M. A. 1995. *Learning Search Control Knowledge to Improve Plan Quality.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. Available as Technical Report CMU-CS-95-175.

Pryor, L. M. 1994. *Opportunities and Planning in an Unpredictable World.* Ph.D. Dissertation, Northwestern University, Evanston, Illinois. Also available as Technical Report number 53.

Simmons, R.; Lin, L.-J.; and Fedor, C. 1990. Autonomous task control for mobile robots. In *Proceedings of the IEEE Symposium on Reactive Control.*

Simmons, R. 1994a. Becoming increasingly reliable. In *Proceedings of AIPS-94*, 152–157.

Simmons, R. 1994b. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation* 10(1).

Stone, P., and Veloso, M. 1995. User-guided interleaving of planning and execution. In *Proceedings of the European Workshop on Planning.*

Tambe, M.; Schwamb, K.; and Rosenbloom, P. S. 1995. Building intelligent pilots for simulated rotary wing aircraft. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation.*

Veloso, M. M.; Carbonell, J.; Pérez, M. A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7(1).

Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning.* Springer Verlag. (Monograph of Ph.D. thesis, Carnegie Mellon University, 1992).

Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of ML-95.*