Planning by Adaptation: Experimental Results

D. M. Lyons
and A. J. Hendriks
Philips Laboratories, Briarcliff Manor NY 10510

Abstract

A robot system operating in an environment in which there is uncertainty and change needs to combine the ability to react with the ability to plan ahead. In a previous paper we proposed a solution to the problems of integrating planning and reaction: cast planning as *adaptation* of a reactive system. In this paper, we present our first experimental results from the planner-reactor architecture.

Introduction

The importance of integrating deliberative ("planning") capabilities and reactive capabilities when building robust, 'real-world' robot systems is becoming widely accepted [Bresina and Drummond, 1990, Fraichard and Laugier, 1991, McDermott, 1991]. In [Lyons *et al.*, 1991] we proposed a solution to this integration: cast planning as *adaptation* of a reactive system. In subsequent papers, we looked at the theoretical issues of incremental adaptation [Lyons and Hendriks, 1992] and of safely making changes to an operating reactive system [Lyons and Hendriks, 1993]. Here we present our first performance results, using our approach to construct a robot kitting workcell.

We begin with a brief review of our application domain and the necessity for developing our approach. Section is a summary of the planner-reactor architecture we developed to implement of our approach. Section describes the kitting robot domain. Sections and provide more details of our implementation as a prelude to our experimental results in Section .

Background

Purely deliberative approaches (e.g., [Fikes and Nilsson, 1971]) are good at determining globally good courses of action for a robot, selecting an optimal assembly sequence for example. They are not very robust, however, in the face of changing or uncertain conditions in the robot's environment. Typically, a deliberative system expends a lot of effort deriving a good course of action for the robot, only to find on starting that the environment has changed and rendered the actions unsuitable [McDermott, 1991]. Purely reactive approaches (e.g., [Brooks, 1986, Gat, 1991]) operate very robustly in some uncertain and dynamic environments — namely those for which the reactions have been manually pre-programmed – but lack any ability to plan ahead. However, many application domains such as mobile robots[Fraichard and Laugier, 1991], autonomous explorers[Bresina and Drummond, 1990], emergency response planning [McDermott, 1991] as well as our domain, robotic kitting, demand the integration of deliberative and reactive capabilities.

A kitting robot [Sellers and Nof, 1989] is a robot system that puts together assembly kits — trays containing all the necessary parts from which to build a specific product. A reactive component allows the kitting robot to handle the uncertainty in the quality and quantity of the raw materials stream and downstream automation [Fox and Kempf, 1985]. However, a deliberative component is still necessary to 'program' the robot to handle different kits/parts or mixes of kits, as well as to implement temporary strategies in the face of disturbances on the factory floor. In [Lyons and Hendriks, 1992] we suggested a straightforward approach to this integration: allow the deliberative component to incrementally 'tune' the reactive component to suit current goals and operating conditions.

Some alternative approaches to this integration have been suggested. Schoppers[Schoppers, 1989] amongst others has proposed an 'off-line' generator for reactive systems; Xiaodong and Bekey [Xiaodong and Bekey, 1988] also suggest something similar for an assembly cell equipped with a reactive scheduler. This approach works well only when the deliberative component can be separated 'off-line'. We address the problems that arise when both reactive and deliberative components need to be 'on-line'. Connell's

SSS [Connell, 1992] addresses this integration for mobile robots. In SSS, though, the deliberative component is restricted to enabling/disabling behaviors, while we need to be able also to generate new behaviors. Bresina & Drummond's ERE [Bresina and Drummond, 1990] comes closest to our approach. A key difference is that our domain, kitting, is a *repetitive* activity. Our concept of the improvement of a reactive system exploits this repetition, whereas ERE was designed for once-off activities.

The Planner-Reactor Architecture

Our definition of planner is a system that continually modifies a concurrent and separate reactive system (the *reactor*) so that its behavior becomes more goal directed. Figure 1 illustrates this architecture.

Figure 1: The Planner-Reactor Architecture.

The reactor contains a network of reactions: hardwired compositions of sensor and motor processes. The key property of the reactor is that it can produce action at *any* time. Unlike a plan executor or a hierarchical system of planner and reactive system, our reactor acts *asynchronously* and *independently* of the planner. It is always actively inspecting the world, and will act should one of its reactions be triggered. A reactor should produce timely, useful behavior even without a planner.

Rather than viewing the planner as a higher-level system that loads plans into an executor, we see the planner as an equal-level system that continually tunes the reactor to produce appropriate behavior (Figure 1). The interaction between the planner and reactor is entirely *asynchronous.* The planner continually determines if the reactor's responses to the current environment would indeed conform to the objectives. If not, then the planner makes an incremental change to the reactor configuration to bring the reactor's behavior more into line with the objectives.

In this section, we briefly reiterate the details of the planner-reactor approach as presented in our previous papers [Lyons and Hendriks, 1992, Lyons and Hendriks, 1993]. We begin by introducing the formal language we use to describe the reactor and its incremental adaptation, $\mathcal{RS}$.

The $\mathcal{RS}$ Model

The $\mathcal{RS}$ model [Lyons, 1993] was developed to represent and analyze the kind of programs involved in sensory-based robotics. This language allows a user to construct programs by 'gluing' together processes in various ways. This style of language is called a *process-algebra* language.

A description of a process, or network of processes, is called a *schema*. For example, $\mathtt{Locate}_m\langle x\rangle$ denotes a process that is an instance of the schema `Locate` with one ingoing parameter m and one outcoming result x. Networks are built by composing processes together using several kinds of *process composition operators*. This allows processes to be ordered in various ways, including concurrent, conditional and iterative orderings. At the bottom of this hierarchy, every network must be composed from a set of atomic, pre-defined processes. The set of *basic schemas* defines what processes are atomic. The composition operations are

- *sequential* `A;B` , do `A` then `B`.
- *concurrent* `A|B`, do `A` and `B` concurrently.
- *conditional* `A:B`, do `B` only if `A` succeeds.
- *negation* `~A`, do `A` but fail if `A` succeeds and vice-versa.
- *disabling* `A#B`, do `A` and `B` concurrently until one terminates.

We use two recurrent operators:

- *synchronous recurrent* $\mathtt{A} :; \mathtt{B} = \mathtt{A} : (\mathtt{B}; (\mathtt{A} :; \mathtt{B}))$ while `A` succeeds do `B`.
- *asynchronous recurrent* $\mathtt{A} :: \mathtt{B} = \mathtt{A} : (\mathtt{B} \mid (\mathtt{A} :: \mathtt{B}))$ while `A` succeeds spawn off `B`.

It is important to represent how process networks evolve over time, as processes dynamically terminate or are created. This is captured by the *evolves* operator: We say that process `A` evolves into process `B` under condition Ω if `A` possibly becomes equal to `B` when condition Ω occurs; we write this as $\mathtt{A} \xrightarrow{\Omega} \mathtt{B}$. A process *must* evolve to one of the set of processes to which it can possibly evolve.

Incremental Reactor Adaptation

We define the reactor `R` as a set of concurrent reactions, and a well-defined interface through which the planner can modify the reactor structure:

$$\mathtt{R} = \quad \Big|_i \ (\mathtt{E}^i \# \mathtt{P}^i) \ \mid \ \mathtt{ADD}\langle k\rangle :: (\mathtt{E}^k \# \mathtt{P}^k) \qquad (1)$$

where there are three main items:

1. $\mathtt{P}^i$ is a reaction or reactor segment
2. $\mathtt{E}^i$ is a guard process by which the planner can remove (by the #-composition) $\mathtt{P}^i$ from the reactor.

3. ADD is the interface through which the planner can add (by the ::-composition) new guarded reactions.

A reactor does not contain/execute plans in the same sense that a classical plan executor does, i.e., as a data structure. The reactions are represented *procedurally,* so the reactor is more like a compiled controller. Nonetheless, certain parts of the reactor structure can be identified as corresponding to the instructions necessary for a specific task or subtask; we call these parts *reactor segments.*

It is possible to write down how the reactor is affected by the changes described in items 2 and 3 of eq. (1) above using the $\mathcal{RS}$ evolves operator:

$$\begin{array}{ccc} (\ \mathtt{R} \mid (\mathtt{E}^k \# \mathtt{P}^k)\ ) & \xrightarrow{Dk} & \mathtt{R} \\ \mathtt{R} & \xrightarrow{Ak} & (\ \mathtt{R} \mid (\mathtt{E}^k \# \mathtt{P}^k)\ ) \end{array} \quad (2)$$

$$\begin{array}{l} where\ Dk\ is\ \mathtt{E}^k \longrightarrow \mathtt{STOP} \\ where\ Ak\ is\ \mathtt{ADD}\langle k \rangle \longrightarrow \mathtt{STOP} \end{array}$$

Saying that a process P successfully terminates (under some condition) is the same as saying that P evolves to STOP (under that condition). So in (2) above, Dk can be used to *delete* reactions, and Ak can be used to *add* reactions. Using the evolves operation, we can express the process of reactor adaptation as follows:

$$\mathtt{R} \xrightarrow{\alpha} \mathtt{R}' \quad (3)$$

where $\mathtt{R}'$ is the adapted reactor and α is some combination of Ak and Dk conditions. In [Lyons and Hendriks, 1993] we describe how to make these adaptations in a safe manner despite the fact that the planner does know when or which reactions might be firing.

Ideal Reactor. The ultimate objective of the planner is the construction of a reactor that achieves its goals not only in the current environment, but because change is possible, in all environments that might occur, as indicated by the environment model. A reactor that fulfills this criterion is called an *ideal reactor*, and is similar in behavior to Schoppers' universal plans[Schoppers, 1989]. It is unrealistic to suppose that a planner will always be able to generate the ideal reactor in one step, and in [Lyons and Hendriks, 1992] we introduced the more restricted ω-ideal reactor $\mathtt{R}^{\omega}$. In planner terms, ω is a set of *assumptions* under which the reactor $\mathtt{R}^{\omega}$ has been constructed. We propose the following incremental reactor construction strategy: ω is initially chosen to allow the planner to quickly produce a working reactor and then ω is gradually relaxed over time. The evolution in (3) can be used to represent this strategy as follows:

$$\begin{array}{c} \mathtt{R}^{\omega 1} \xrightarrow{\alpha 1} \mathtt{R}^{\omega 2} \xrightarrow{\alpha 2} \mathtt{R}^{\omega 3} \xrightarrow{\alpha 3} \ldots \xrightarrow{\alpha n} \mathtt{R}^{*} \\ for\ \ e(\omega 1) > e(\omega 2) > e(\omega 3) > \ldots > e(\emptyset) \end{array} \quad (4)$$

where the αi are the adaptation commands generated by the planner and $e()$ is an evaluation function. The assumption relaxation priorities are governed by the ordering on ω dictated by $e()$. The conditions that trigger the planner to produce an improved reactor are described in [Lyons and Hendriks, 1992] and the constraints on convergence to $\mathtt{R}^*$ overviewed.

- $\mathtt{ASSERT}_{s,p,\ldots}$ Assert situation s and initialize the situation parameters to $p,\ldots$. This terminates when the situation is terminated and stops or aborts depending on whether FAIL or SUCCEED was used.
- $\mathtt{FAIL}_{s,k}$ Terminates instance k of situation s with fail status.
- $\mathtt{SUCCEED}_{s,k}$ Terminates instance k of situation s with success status.
- $\mathtt{SIT}_s\langle k, p1, p2, \ldots \rangle$ Terminates if an instance of situation s is currently asserted. k is the instance number and $p1, p2, \ldots$ are the current values of the situation parameters.
- $\mathtt{REASSERT}_{s,p,\ldots}$ Assert s and whenever it ends with fail status reassert it.

Table 1: Situation basic processes.

The Kitting Robot Problem

Assembly components are fed from stores to a kitting station on a conveyor belt. The kitting station consists of one or more kitting robots. Each kitting robot must take parts off the belt, place them in the appropriate slots of a kitting tray, and then place the kitted tray onto the belt. The trays are stored in a stack in the robot's workspace. The product we are kitting is a small DC servomotor manufactured by Philips. It has three parts: a cap, a motor and a body, all of which have a number of variants.

The planner-reactor approach provides a way to incrementally construct reactive systems with improving performance. At the heart of this iterative mechanism is the concept of characterizing the environment by a set of assumptions and methods to detect whether they hold or not.

Kitting Assumptions. The set of assumptions we have developed for the kitting environment is described below in reverse order of *likelihood of holding*: the ordering we have chosen for the planner to relax them (i.e., the $e()$ function in (4) above). The planner uses these assumptions to help it to quickly build a reasonable reactor. Later, as it has time, it relaxes assumptions and fashions improved versions of the reactor. In the case that the environment goes through regimes where different subsets of assumptions hold, the planner can capitalize on this, especially when new factory goals have been received and resultant changes in behavior are necessary.

Some of the assumptions, such as that of the quality and substitutability of parts, *must* eventually be relaxed to get to a robust workcell. Some other assumptions, such as that of cooperation in kit assembly and tray disturbance, describe contingencies that make a more versatile workcell, but are not necessary for a robust system. Still other assumptions pertain to unusual operating conditions, such as the assumptions of no competition and no downstream disturbances.

1. Assumption of Parts Quality (AQ): All the parts coming into the kitting workcell are of good quality and do not need to be tested.
2. Assumption of non-substitutability of parts (AS): Each part has only one variant.
3. Assumption of unfilled trays (AF): All trays arrive empty.
4. Assumption of no competition (ACT): Parts placed into a tray stay there.
5. Assumption of no cooperation (ACP): The robot assumes that it is not sharing the job of populating kits with other workcells.
6. Assumption of no tray disturbance (ADT): Trays remain where they were put.
7. Assumption of no parts motion (AM): The kit parts do not move around once delivered on the belt to the workcell.
8. Assumption of no downstream disturbance (ADD): Downstream automation is always ready to receive finished kits.
9. Assumption of parts ordering (AO): The robot cannot modify the preplanned parts ordering.

In addition to its normal relaxation sequence, the planner can also be *forced* to relax assumptions. Whenever the planner builds a reactor that depends upon a particular assumption holding, it needs to embed a monitor process that can detect whether the assumption actually holds in practice or not. The monitor process triggers a 'clean up' action if the assumption ever fails, and notifies the planner that it needs to relax this assumption next.

Situations

Reactor situations are a mechanism to group related reactions together in a hierarchical fashion. This modularity is important because it allows (human) designers to more easily understand the reactor and diagnose any problems that may occur; it provides the planner with a unit around which to define safe changes to the reactor structure [Lyons and Hendriks, 1993]; and it provides a way to give computational resources to those reactions that currently need it, while 'suspending' others.

Intuitively: a situation being active expresses the appropriateness for the reactor to enable the set of reactions associated with that situation. Situations can be nested hierarchically, and many situations may be active at one time so that the execution of their reactions will be interleaved. We introduce a small set of basic processes with which to build reactor situations in $\mathcal{RS}$ (table 1). For example, an instance of a situation is asserted by the execution of the $\mathtt{ASSERT}_{s,p1,p2,\ldots}$ process, where $p1, p2, \ldots$ are the values for the parameters associated with the situation. The $\mathtt{SIT}_s\langle k, p1, \ldots\rangle$ process, when executed, suspends itself until an instance of situation s is asserted. It then terminates and passes on the details of the situation instance as its results.

We use the $\mathtt{SIT}_s$ process to ensure that a reaction associated with a situation is only 'enabled' when an instance of that situation is asserted. For example, let $\mathtt{P}^1, \ldots, \mathtt{P}^n$ be the reactions for situation s, then we would represent this in the reactor as

$$\mathtt{PS} = \mathtt{SIT}_s :; \mathtt{P}^1 \mid \mathtt{SIT}_s :; \mathtt{P}^2 \mid \ldots \mid \mathtt{SIT}_s :; \mathtt{P}^n$$

The ':;' operation ensures that as long as the situation is active, then the reactions are continually re-enabled. Note that once enabled, a reaction cannot be disabled until it has terminated. The asserting and termination of instances of situation s happen as the side effects of $\mathtt{ASSERT}_s$ and $\mathtt{FAIL}_s$ or $\mathtt{SUCCEED}_s$ processes in reactions in other situations.

Planning as Adaptation

This section outlines the principles of operation of the planner, a detailed description of its architecture is given in [Lyons and Hendriks, 1992].

Adaptation Increments

At every planner iteration t, the planner generates what we call an *expectation*, E_t; an abstract description of the changes it expects to make to the reactor to achieve the current goals G_t. The combination of the reactor model R_t and the expectation is always the ω-ideal reactor (where ω is the current set of assumptions the planner is working with).

To reduce this expectation, the planner reasons within a Problem Solving Context (PSC) consisting of the relevant parts of the environment model, the action repertoire and the assumptions ω. The outcome will be a reactor adaptation ΔR_t, the reactions necessary to implement the expectation reduction ΔE_t. Given a set of goals, the initial expectation reduction will be the construction of an an abstract plan. This abstract plan is incrementally transferred into the reactor by the insertion of situations, and is also used for search control in the planner itself. This has two advantages: firstly it allows the planner to adapt the reactor partially, with some of the reactor segments being a STUB, i.e., a process that does nothing except notify the planner that this particular segment has become active. Secondly it allows modular refinements due to, e.g., assumption relaxation later on.

For the kitting robot domain, an initial abstract plan would be a sequential composition of the situations $AcquireTray$, $FillTray$, and $RemoveTray$. Only $AcquireTray$ needs to be concrete in order for the planner to adapt the reactor. The other situations can have STUB reflexes initially. The planner can refine these while the reactor acquires a tray.

The process descriptions below show a reactor segment generated by the planner for this initial adaptation. Each reaction in the segment below is in an if-then-else form (˜(Cond : IfTrue) : Else). The REASSERT process continually reasserts failing situations. P0 is necessary to assert the topmost situation repeatedly. The process $\mathtt{Move}_{obj,loc}$ repositions a grasped object obj to a location loc.

$$
\begin{aligned}
\mathtt{P0} &= \mathtt{STOP} :; \mathtt{ASSERT}_{Kitting} \\
\mathtt{P1} &= \mathtt{SIT}_{Kitting}\langle k\rangle \;:;\; (\\
&\quad \tilde{}((\mathtt{REASSERT}_{AcquireTray}\langle x1\rangle : \mathtt{REASSERT}_{FillTray,x1} \\
&\qquad : \mathtt{REASSERT}_{FinishTray,x1}) : \mathtt{SUCCEED}_{Kitting,k}) \\
&\qquad\quad : \mathtt{FAIL}_{Kitting,k}\ ) \\
\mathtt{P2} &= \mathtt{SIT}_{AcquireTray}\langle k\rangle \;:;\; (\\
&\quad \tilde{}(\mathtt{REASSERT}_{FindPart}\langle x1\rangle : \mathtt{Move}_{x1,trayarea} \\
&\qquad : (\mathtt{UPDATE}_{AcquireTray,k,x1}; \mathtt{SUCCEED}_{AcquireTray,k})) \\
&\qquad\quad : \mathtt{FAIL}_{AcquireTray,k}\ ) \\
\mathtt{P5} &= \mathtt{SIT}_{FinishTray}\langle k, x1\rangle \;:; \\
&\quad (\mathtt{STUB}_{FinishTray} : \mathtt{SUSPEND}_{2000}; \mathtt{FAIL}_{FinishTray,k})
\end{aligned}
$$

Barring any perceptions received from the reactor, the planner will continue to incrementally flesh out abstract segments and adapt the reactor, until all the reactor segments are made concrete, i.e., all the STUB processes removed. The planner has then (by definition) achieved an ω-ideal reactor, and can proceed to relax the next assumption. If any segment relies on an assumption then the adaptation will additionally contain an assumption monitor for that assumption.

Forced Assumption Relaxation.

Any of the assumption monitors or stub triggers in the initial reactor segments can potentially signal the planner and divert its attention from the a-priori established ordering of assumption relaxation. Failure of a situation to successfully complete its reactions is also cause for a perception. These three sources of perceptual input have different effects on the planner.

On receiving a stub trigger perception, the planner redirects its focus of attention to that portion of the plan containing the stub trigger. An assumption failure perception has a larger effect than simply refocusing attention. This perception causes the planner to negate the assumption in its PSC and begin to rebuild the effected portions of its plan. If a situation has failed that relied on that assumption, that particular situation is given highest priority for adaptation. This may cause the planner to refocus on parts of the plan it had previously considered finished.

Apart from its cause, a forced relaxation or refinement results in the same reactor segment and adaptation sequence as a normal relaxation or refinement. Once the planner has achieved a complete ω-ideal reactor, it is ready again to select the next assumption for normal relaxation.

Experimental Results

The kitting example described section was implemented and several experimental runs conducted on a PUMA-560 based kitting workcell. Trace statistics were gathered on each run on, e.g., the times when assumption failures were detected, when adaptations were made, the number of assumptions in use in the reactor, and so on. In each run, the planner utilized all the assumptions described in section . However, only five of the assumptions were actually relaxed in these trials. The purpose of these experimental runs was to begin to explore the behavior of a planner-reactor system in practice. Results from the runs are presented in the following subsections.

Initial Reactor Construction. Figure 2 contains trace statistics for the startup phase on a typical experimental run of the kitting workcell. Graph (a) shows the number of assumptions in use at any time in the workcell by the planner and by the reactor. Graph (b) shows the number of *adaptations issued* (by the planner) and the number of discrete changes to the reactor structure (*adaptations applied*). Initially there is an empty reactor in operation. After 26 seconds (in this example) the planner is in a position to start to update the reactor. (This time include planning plus communication time.)

The planner begins by sending adaptations to the reactor (*adaptations issued*). Each adaptation involves one or more changes to the reactor structure. The constraints involved in safe adaptation can cause a time lag in implementing changes. Active situations cannot be interrupted — that would leave the reactor in an undefined state — instead the adaptation waits for the situation to end before applying the changes; the theory behind this is presented in [Lyons and Hendriks, 1993]. Thus the *adaptations applied* trace always lags the *adaptations issued* trace. The lag time varies, depending on the activity of the reactor. This is what causes the reactor assumptions trace in fig. 2(a) to lag the planner assumption trace.

The first reactor is in place roughly 28s after startup.

Figure 2: Startup phase statistics for one run of the Planner-Reactor based Kitting Robot.

Even though this reactor is not complete, the kitting robot can now start kitting; its kitting actions are overlapped in time with the further refinement of its kitting 'program'. This reactor is elaborated over the following 20s until by 40s after startup the first complete reactor is in place. This reactor employs all nine assumptions. Up to this point, there has been little delay between the planner reducing an assumption and the reactor following suit.

Assumption Relaxation. Now the planner begins to relax assumptions to incrementally improve the reactor. It takes roughly 5s to relax the first assumption (the length of the 'plateau' in the planner trace in figure 2(a)). It then begins to issue the adaptations to the reactor. Typically one or more adaptations is necessary to relax any one assumption. Each adaptation will give rise to one or more structural changes in the reactor, again subject to the delays of the safe adaptation constraint. In the case of this first assumption relaxation, its takes the reactor roughly 60s before it has been able to implement the last structural change to relax the assumption (the 'plateau' on the reactor trace in fig.2(a)). It takes the planner roughly 30s to relax the second assumption; but the change is implemented in the reactor almost immediately after the first relaxation. These lags and overlaps in operation underscore the highly asynchronous and independent operation of planner and reactor.

In these experiments, we restricted to planner to only relax two assumptions. Having done this, the planner will only relax other assumptions when the environment forces their failure. This could occur within the startup phase, of course, but for convenience of presentation we have separated the two here.

Environment Forced Relaxation. Figure 3 shows trace statistics for the ongoing operational phase in a typical experimental run of the kitting workcell. Again, graph (a) displays the assumptions in use by the planner and reactor, and graph(b) displays the trace of adaptations issued and applied. Additionally, fig.3(b) now also shows the trace of assumption failure perceptions. These are the perceptions that force the planner to relax an assumption. The two forced assumption relaxations in fig.3 are the no-motion (AM) and downstream disturbance (ADD) assumptions. The failure perception for AM is generated when the robot fails to acquire a part it had identified in an initial visual image of the workspace. The failure perception for ADD is manually invoked, and would normally be an input signal from downstream automation.

At 152s the AM assumption failure is signaled (Fig.3(b)). The planner responds very quickly and begins to issue adaptations by 155s, continuing for about 10s. There is little delay in the safe adaptation procedure and the structural changes to the reactor lag the adaptations issued by only about 1s. This is also evident from the assumptions trace in (b). Part of the changes in this adaptation is to remove the assumption failure monitor for this assumption; thus, partway into the adaptation (161s) the signals from the assumption monitor cease. The total forced relaxation response time — the time from failure to final change in the reactor — is roughly 11s. The ADD failure occurs at 215s and proceeds in a similar fashion.

Reinstating Assumptions. In our theoretical analysis [Lyons and Hendriks, 1992, Lyons and Hendriks, 1993] we did *not* address the problem of reasserting previously failed assumptions. Nonetheless, this is convenient in practice to model operating regimes, and therefore is part of our objectives. In the experimental run, at time 357s the ADD assumption was reinstated (fig.3(b)). On the relaxation of such reinstateable assumptions, the planner adds a reactor monitor that will signal when the assumption holds again. (It doesn't show up as an

Figure 3: Forced relaxation statistics.

assumption failure, since it isn't). As the trace statistics show, our implementation does indeed handle the reintroduction of assumptions. However, further theoretical work is necessary to extend our convergence results to include this case.

Discussion

In summary, we have reviewed our motivation and approach to casting planning as adaptation of a reactive system and have presented the first performance results of the approach, a first for integrated systems of this kind.

These results show the feasibility of our approach. The incremental reactor construction and assumption relaxation strategy minimizes planning delays. The reactor can start kitting while the planner still is refining later parts of the task. The asynchronous communication between planner and reactor allows the latter to be real-time and not bogged down by the planner. The safe adaptation algorithm guarantees at all times a correct operation of the reactor, while avoiding interruption of non-affected reactor segments.

Further work. We need to extend our theoretical results to include assumption reinstating to model operating regimes. In conjunction, quantitative comparisons with other approaches to build integrated planning-reacting systems are planned.

References

[Bresina and Drummond, 1990] J. Bresina and M. Drummond. Integrating planning and reaction. In J. Hendler, editor, *AAAI Spring Workshop on Planning in Uncertain, Unpredictable or Changing Environments*, Stanford CA, Mar. 27–29 1990. Systems Research Center, Univ. of Maryland.

[Brooks, 1986] R. Brooks. A robust layered control system for a mobile robot. *IEEE J. Rob. & Aut.*, RA-2(1):14–22, Mar. 1986.

[Connell, 1992] J. Connell. Sss: A hybred architecture applied to robot navigation. In *IEEE Int. Conf. Robotics & Automation*, pages 2719–2724, Nice, France, May 1992.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fox and Kempf, 1985] B.R. Fox and K.G. Kempf. Opportunistic scheduling for robotic assembly. In *IEEE Int. Conf. Robotics & Automation*, pages 880–889, St.Louis MO, 1985.

[Fraichard and Laugier, 1991] Th. Fraichard and C. Laugier. On-line reactive planning for a non-holonomic mobile in a dynamic world. In *IEEE Int. Conf. Robotics & Automation*, pages 432–437, Sacramento, CA, Apr. 1991.

[Gat, 1991] E. Gat. Alfa: A language for programming reactive robotic control systems. In *IEEE Int. Conf. Robotics & Automation*, pages 1116–1121, Sacramento, CA, Apr. 1991.

[Lyons and Hendriks, 1992] D.M. Lyons and A.J. Hendriks. Planning for reactive robot behavior. In *IEEE Int. Conf. Rob. & Aut.*, Apr. 7–12th 1992. (See also video proceedings).

[Lyons and Hendriks, 1993] D.M. Lyons and A.J. Hendriks. Safely adapting a hierarchical reactive system. In *SPIE Intelligent Robots and Computer Vision XII*, September 1993.

[Lyons *et al.*, 1991] D.M. Lyons, A.J. Hendriks, and S. Mehta. Achieving robustness by casting planning as adaptation of a reactive system. In *IEEE Int. Conf. Rob. & Aut.*, Apr. 7–12th 1991.

[Lyons, 1993] D.M. Lyons. Representing and analysing action plans as networks of concurrent processes. *IEEE Trans. Rob. & Aut.*, 9(3), June 1993.

[McDermott, 1991] D. McDermott. Robot planning. Technical Report YALEU/CSD/RR#861, Yale University, Aug. 1991.

[Schoppers, 1989] M. Schoppers. *Representation and Automatic Synthesis of Reaction Plans.* Technical Report UIUCDCS-R-89-1546 (PhD Thesis), Dept of Computer Science, University of Illinois at Urbana-Champaign, 1989.

[Sellers and Nof, 1989] C.J. Sellers and S.Y. Nof. Performance analysis of robotic kitting systems. *Rob. & Comp. Integ. Manuf.*, 6(1):15–24, 1989.

[Xiaodong and Bekey, 1988] X. Xiaodong and G.A. Bekey. Sroma: An adaptive scheduler for robotic assembly systems. In *IEEE Int. Conf. Robotics & Automation*, pages 1282–1287, Philadelphia, PA, 1988.