Adaptive Planning: An Approach that Views Errors as Assumption Failures

D. M. Lyons
and A. J. Hendriks
Philips Laboratories
Philips Electronics, North America Corporation
Briarcliff Manor NY 10510

Abstract

To work effectively in a dynamic and uncertain environment, an automated manufacturing system needs to be able to carry out actions that achieve its objectives, despite the broad range of, possibly unpredictable, events that may occur. Even in such environments, there may be regularities in behavior. These regularities can be embodied as *assumptions*, and used to simplify the construction of a controller for the system. However, at any time, only a subset of all assumptions might hold, and a controller relying on an assumption that does not hold will incurr errors. Therefore, it is necessary to continuously monitor assumptions and to dynamically adapt the controller when the set of working assumptions change.

This paper describes an approach, called the *Planner-Reactor* approach, that exploits knowledge of regularities of the environment to incrementally build, and then continually adapt, a controller that operates effectively in the current vironment. The theory of the planner-reactor approach is briefly described, but the focus of the paper is on experimental results from an implementation of an industrial kitting robot using the approach.

Introduction

To work effectively in a dynamic and uncertain environment, an automated manufacturing system needs to be able to carry out actions that achieve its objectives, despite the broad range of, possibly unpredictable, events that may occur. Our application domain is the kitting robot [Sellers and Nof, 1989]: a robot system that puts together assembly kits — trays containing all the necessary parts from which to build a specific product.

One approach to building a controller for a kitting robot is to build a planner that accepts a description of the kit or kits the robot must fill, and which produces detailed plans for each kit, which can then be executed repeatedly on the robot. The problem with this is that the planner does not know at planning time exactly what the conditions at plan execution time will be. Thus, it may produces plans that will sometimes cause execution errors to occur, or it may produce plans that are 'overkill' for the task. Sometimes it is possible to produce a *universal* plan [Schoppers, 1989] – a plan that will work for every possible combination of conditions. However, often a universal plan may be impossibly large, or take a very long time to produce, or lead to other computational or efficiency problems.

The so-called reactive approaches, e.g., behavior-based work[Brooks, 1986], have proven robust in uncertain and dynamic environments. However, they do not support the deliberation necessary to produce kitting plans. Our primary objective is to achieve the kind of robustness evidenced by reactive approaches, while maintaining the deliberative capability of planning approaches. We do this by, firstly, capturing regularities of the dynamic and changing environment in a set of assumptions that the planner can use to simplify the construction of a reactive kitting cell controller; and, secondly, by having the planner 'monitor' the kitting cell for error reports, which it then uses to iteratively refine the assumptions on which the reactive controller operates.

In our kitting robot application, the reactive controller will allow the robot to handle the uncertainty in the quality and quantity of the raw materials stream and downstream automation. The deliberative component is 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.

The remainder of the paper is as follows. The second section is a summary of the planner-reactor architecture we developed to implement of our approach. The third section describes the kitting robot domain. The next three short sections provide more details of our implementation as a prelude to our experimental results in section seven. A comparison of our approach to similar work in the planning and learning communities is presented in section eight.

The Planner-Reactor Architecture

Our definition of a 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.

The reactor contains a network of reactions — hard-wired compositions of sensor and motor processes. The

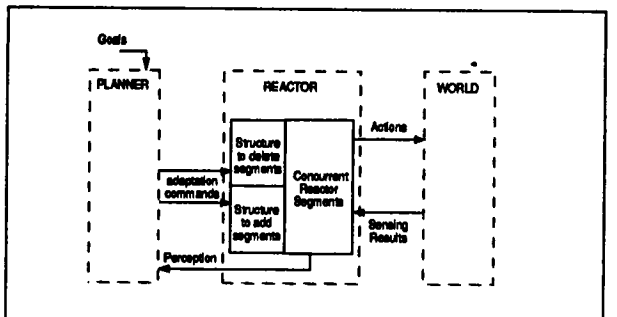

Figure 1: The Planner-Reactor-World System.

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. 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 highlight the most salient aspects of the planner-reactor approach. We refer to our previous work for the mathematical descriptions of the reactor[Lyons and Hendriks, 1992] and its safe adaptation mechanism [Lyons and Hendriks, 1993] in the formal language $\mathcal{RS}$.

Incremental Reactor Adaptation. The reactor R consists of a set of concurrent reactions, and a well-defined interface through which the planner can modify the reactor structure, by either adding or deleting reactions (Figure 1).

A reactor does not contain/execute plans in the same sense that a classical plan executor does, i.e., as a data structure. Rather, 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.*

The reactor contains additional structure to safeguard addition and deletion of reactor segments, to prevent the reactor ending up in an undefined state. Briefly, each segment is tagged with a guard, a special process that can terminate a reactor segment's existence. Other special processes can inhibit reactor segment triggering processes from firing, even when they are enabled. A combination of these two mechanisms allows the planner to insert new reactions that wait for a segment to be not active, inhibits it from becoming active again, before removing it from the reactor. In [Lyons and Hendriks, 1993] we describe this mechanism in detail with emphasis on how to make these adaptations in a safe manner despite the fact that the planner does not 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. Process evolution [Lyons, 1993] can be used to capture this strategy.

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.

The iterative refinement strategy can be captured using process evolution as follows:

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

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 $e()$.

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. Typically, the failure of some action within the reactor will be the signal that an assumption has been violated. Strictly speaking, it is necessary to go through a *diagnosis* step (e.g., [Kumaradjaja and DiCesare, 1988]) to determine from observed errors *which* assumption has been violated.

Our emphasis is not on diagnosis, so we simplify that aspect of the problem here to a simple, known mapping between observed errors and assumption failures.

The Kitting Robot Problem

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. 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 (1) 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 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 assumption of no downstream disturbances. The following are some of the (nine in total) assumption used in our implementation:

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 no parts motion (AM): The kit parts do not move around once delivered on the belt to the workcell.
4. Assumption of no downstream disturbance (ADD): Downstream automation is always ready to receive finished kits.

The $\mathcal{RS}$ Model

We employ a language called $\mathcal{RS}$ to represent and analyze the planner-reactor framework. $\mathcal{RS}$ [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. $\mathcal{RS}$ has the additional advantage of being a *dynamic* model: thus it simplifies the representation of dynamically adding in or removing controller structure, as the planner needs to do in our approach. In static computational models, such as automata or Petri-Net models, this is very difficult to represent. The $\mathcal{RS}$ language is described extensively in [Lyons, 1993] and here we include only a short introduction.

In $\mathcal{RS}$ notation, $\mathtt{P}_m\langle x\rangle$ denotes a process that is an instance of the schema P with one ingoing parameter m and one outcoming result x. Process 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 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* $\mathtt{A} :; \mathtt{B} \triangleq \mathtt{A} : (\mathtt{B}; (\mathtt{A} :; \mathtt{B}))$ while A succeeds do B.
- *asynchronous recurrent* $\mathtt{A} :: \mathtt{B} \triangleq \mathtt{A} : (\mathtt{B} \mid (\mathtt{A} :: \mathtt{B}))$ while A succeeds spawn off B.

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 $\mathcal{RS}$. 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.

Planning as Adaptation

This section outlines the principles of operation of the planner, a detailed description of its architecture and examples of the system in operation are 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.

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 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 the kitting robot 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. 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 reac-

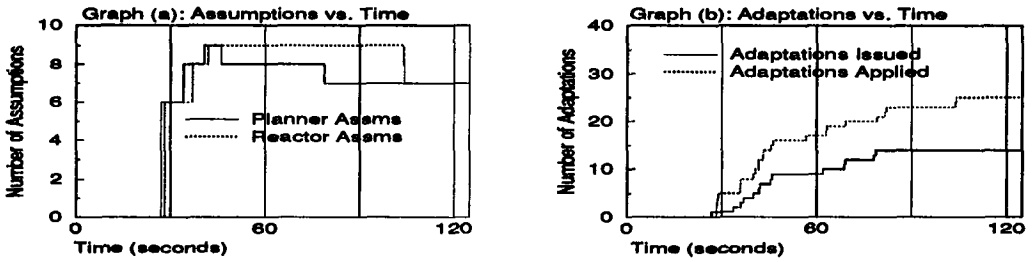

Figure 2: Startup phase statistics for one run of the Planner-Reactor based Kitting Robot.

tor. 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 reinstatable assumptions, the planner adds a reactor monitor that will signal when the assumption holds again. (It doesn't show up as an 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.

Performance Improvement. The essence of the planner-reactor approach is that the reactor is being continually adapted so as to improve its performance. Ideally, to evaluate an implementation, an empirical measurement that quantifies performance should be collected for the duration of an experimental run. The direction and rate of performance improvement can then be directly evaluated. An example of such a performance measurement would be the number of situation or action failures reported. Unfortunately, in a dynamic and uncertain environment, simply because an action *may* fail, does not mean it *will* fail in any given trial. Thus, such a performance measurement must be made over a statistically significant number of trials. There are practical difficulties for us in collecting this amount of trials on our PUMA testbed.

It is straightforward, on the other hand, to measure the ongoing introduction and relaxation of assumptions in the reactor in response to new goals and to the environment. An indirect performance improvement measure can be formulated based on these, and used to illustrate the ongoing improvement. Figure 4 shows a graph of such a measurement for the same experimental run described in graphs 2 and 3. The performance

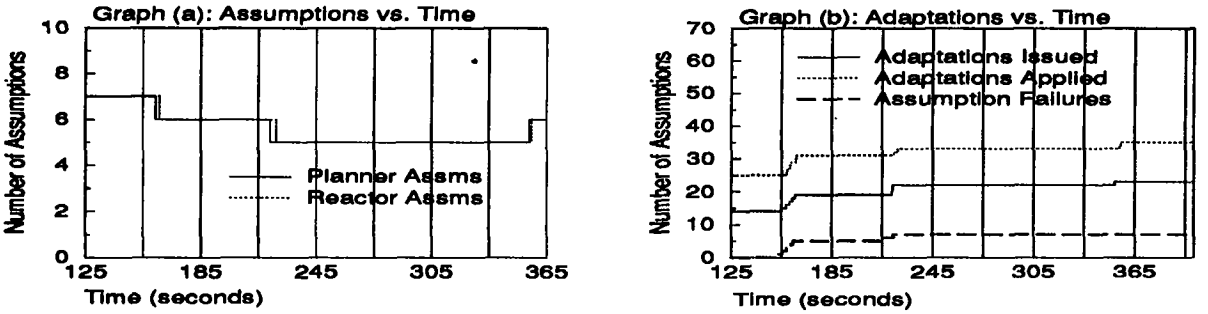

Figure 3: Forced relaxation statistics.

measurement is graphed as $\frac{1}{M}$ versus time, where M has three components:

- *Supported Assumptions:* Let $\#SAP$ be the number of assumptions used in the planner that *also* hold in the environment. Let $\#SAR$ be the same measurement for the reactor. The first component of M is $\#SAP+\#SAR$, a measure of how well the system is supported by the environment.
- *Retracted Assumptions:* Let $\#RAP$ be the number of assumptions that the planner has retracted *to date* in its deliberations. Let $\#RAR$ be the number that have actually been retracted *to date* in the reactor. The second component of M is $\#RAP + \#RAR$, a measure of how much progress the planner has made.
- *Environment Assumption Changes:* Let $\#AE_t$ be the number of assumptions that hold in the environment at time t. The final component of M is $\#AE_{t-1} - \#AE_t$ a measure of the environment changing.

Since retractions affect the first component of M *as well as* the second, its necessary to multiply the second component by 2 to 'cancel' out the effect on the first. Overall M will is bigger for systems that have more supported assumptions and more assumptions retracted – an improved system. It is necessary to add in the change of assumptions in the environment as a 'penalty' so that an assumption change in the environment will register, and the measurement will indicate an 'inferior' system. To graph this measurement in the typical form of a 'learning curve', the reciprocal is taken.

The overall trend in figure 4 is clear: the performance measure decreases over time. The initial decrease comes from the initial construction and then planner-directed assumption relaxation of the reactor. The 'setback' in improvement at times at 152s and 215s are due to assumption failures. The 'setback' at time 357s is not an assumption failure. It is due rather to the fact that the ADD assumption has been reintroduced in the environment but not yet exploited by the system.

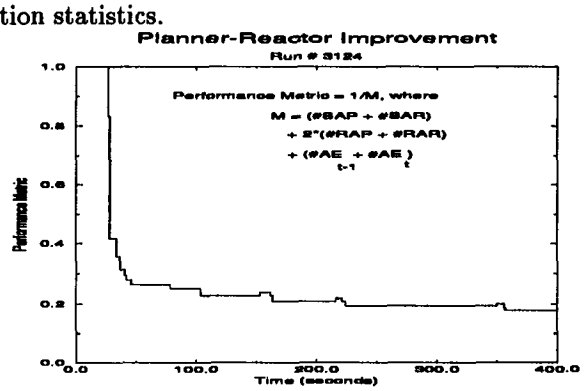

Figure 4: Planner-Reactor Improvement Graph.

Discussion

Some other approaches to integrating deliberative and reactive components in a single system 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 behavioral constraint strategy in ERE and our abstract plan are comparable. We chose however to retain the plan structure in the reactor, have all conflict resolution explicit, and have developed explicit 'surgery' procedures to retract or modify reactor segments in a safe manner. In our system, sensing requires conscious actions and must be planned for. The reactor can start new actions asynchronously, and multiple actions may be in progress at one time. In contrast, ERE operates with a select-execute-sense cycle, where an hidden sensory subsystem keeps track of all information, even if it were irrelevant for the current action selection. The incremental strategies also differ in that

we use the reactor to guide the planner to the relevant parts of the plan to elaborate.

A number of systems in the learning literature have an architecture broadly similar to our planner-reactor system: MD [Chien *et al.*, 1991], Dyna [Sutton, 1990], and SEPIA [Segre and Turney, 1992], amongst others. The viewpoint from which we designed the planner-reactor system is that there is no learning going on: The planner, which knows implicitly all reactors up to and including the ideal reactor, is simply choosing to express different reactors depending on its perception of the environment. If, on the other hand, the planner's world model was refined over time (as, e.g., in [Kedar *et al.*, 1991]), then, we would have argued, there is learning going on.

Conclusions

In summary, we have briefly reviewed our motivation and approach to casting planning as adaptation of a reactive system. This approach calls for treating errors as assumption failure signals. We then resented the first performance results for the approach, a first also 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.

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.

[Chien *et al.*, 1991] S. Chien, M. Gervasion, and G. DeJong. On becoming decreasingly reactive: Learning to deliberate minimally. In *9th Machine Learning Workshop*, pages 288–292, 1991.

[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.

[Kedar *et al.*, 1991] S. Kedar, J. Bresina, and C. Dent. The blind leading the blind: Mutual refinement of approximate theories. In *9th Machine Learning Workshop*, pages 308–312, 1991.

[Kumaradjaja and DiCesare, 1988] R. Kumaradjaja and F. DiCesare. A causal reasoning approach for planning error recovery in automated manufacturing systems. In *SPIE Symp. on Advances in Intelligent Robotic Systems; Expert Robots for Industrial Use (Vol. 1008)*, pages 144–153, 1988.

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

[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.

[Segre and Turney, 1992] A. Segre and J. Turney. Sepia: A resource-bounded adaptive agent. In J. Hendler, editor, *First Int. Conf. on AI Planning Systems*, pages 303–304. Morgan-Kaufman, 1992.

[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.

[Sutton, 1990] R. Sutton. First results with Dyna. In J. Hendler, editor, *AAAI Spring Symposium on Planning in uncertain and changing environments*, Stanford CA, Mar. 27–29 1990. Systems Research Center, Univ. of Maryland.

[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.