Becoming Increasingly Reactive

Tom M. Mitchell

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
Tom.Mitchell@cs.cmu.edu

Abstract

We describe a robot control architecture which combines a stimulus-response subsystem for rapid reaction, with a search-based planner for handling unanticipated situations. The robot agent continually chooses which action it is to perform, using the stimulus-response subsystem when possible, and falling back on the planning subsystem when necessary. Whenever it is forced to plan, it applies an explanation-based learning mechanism to formulate a new stimulus-response rule to cover this new situation and others similar to it. With experience, the agent becomes increasingly reactive as its learning component acquires new stimulus-response rules that eliminate the need for planning in similar subsequent situations. This Theo-Agent architecture is described, and results are presented demonstrating its ability to reduce routine reaction time for a simple mobile robot from minutes to under a second.

1. Introduction and Motivation

Much attention has focused recently on *reactive* architectures for robotic agents that continually sense their environment and compute appropriate reactions to their sense stimuli within bounded time (e.g., (Brooks, 1986, Agre and Chapman, 1987, Rosenschein, 1985)). Such architectures offer advantages over more traditional open-loop search-based planning systems because they can react more quickly to changes to their environment, and because they can operate more robustly in worlds that are difficult to model in advance. *Search-based* planning architectures, on the other hand, offer the advantage of more general-purpose (if slower) problem solving mechanisms which provide the flexibility to deal with a more diverse set of unanticipated goals and situations.

This paper considers the question of how to combine the benefits of reactive and search-based architectures for controlling autonomous agents. We describe the Theo-Agent architecture, which incorporates both a reactive component and a search-based planning component. The fundamental design principle of the Theo-Agent is that it reacts when it can, plans when it must, and learns by augmenting its reactive component whenever it is forced to plan. When used to control a laboratory mobile robot, the Theo-Agent in simple cases learns to reduce its reaction time for new tasks from several minutes to less than a second.

The research reported here is part of our larger effort toward developing a general-purpose learning robot architecture, and builds on earlier work described in (Blythe and Mitchell, 1989). We believe that in order to become increasingly successful, a learning robot will have to incorporate several types of learning:

- It must become *increasingly correct* at predicting the effects of its actions in the world.
- It must become *increasingly reactive*, by reducing the time required for it to make rational choices; that is, the time required to choose actions consistent with the above predictions and its goals.
- It must become *increasingly perceptive* at distinguishing those features of its world that impact its success.

This paper focuses on the second of these types of learning. We describe how the Theo-Agent increases the scope of situations for which it can quickly make rational decisions, by adding new stimulus-response rules whenever it is forced to plan for a situation outside the current scope of its reactive component. Its explanation-based learning mechanism produces rules that recommend precisely the same action as recommended by the slower planner, in exactly those situations in which the same plan rationale would apply. However, the learned rules infer the desired action immediately from the input sense data in a single inference step--without considering explicitly the robot's goals, available actions, or their predicted consequences.

1.1. Related Work

There has been a great deal of recent work on architectures for robot control which continually sense the environment and operate in bounded time (e.g., (Brooks, 1986, Schoppers, 1987, Agre and Chapman, 1987)), though this type of work has not directly addressed issues of learning. Segre's ARMS system (Segre, 1988) applies explanation-based learning to acquire planning tactics for a simulated hand-eye system, and Laird's RoboSoar (Laird and Rosenbloom, 1990) has been applied to simple problems in a real hand-eye robot system. While these researchers share our goal of developing systems that are increasingly reactive, the underlying architectures vary significantly in the form of the knowledge being learned, underlying representations, and real response time. Sutton has proposed an inductive approach to acquiring robot control strategies, in his DYNA system (Sutton, 1990), and Pommerleau has developed a connectionist system which learns to control an outdoor road-following vehicle (Pommerleau, 1989). In addition to work on learning such robot control strategies, there has been much recent interest in robot learning more generally, including work on learning increasingly correct models of actions (Christiansen, et al., 1990, Zrimic and Mowforth, 1988), and work on becoming increasingly perceptive (Tan, 1990).

The work reported here is also somewhat related to recent ideas for compiling low-level reactive systems from high-level specifications (e.g., (Rosenschein, 1985)). In particular, such compilation transforms input descriptions of actions and goals into effective control strategies, using transformations similar to those achieved by explanation-based learning in the Theo-Agent. The main difference between such design-time compilation and the explanation-based learning used in the Theo-Agent, is that for the Theo-Agent learning occurs incrementally and spread across the lifetime of the agent, so that the compilation transformation is incrementally focused by the worlds actually encountered by the agent, and may be interleaved with other learning mechanisms which improve the agent's models of its actions.

The next section of this paper describes the Theo-Agent architecture in greater detail. The subsequent section presents an example of its use in controlling a simple mobile robot, the learning mechanism for acquiring new stimulus-response rules, and timing data showing the effect of caching and rule learning on system reaction time. The final section summarizes some of the lessons of this work, including features and bugs in the current design of the architecture.

2. The Theo-Agent Architecture

The design of the Theo-Agent architecture is primarily driven by the goal of combining the complementary advantages of reactive and search-based systems. Reactive systems offer the advantage of quick response. Search-based planners offer the advantage of broad scope for handling a more diverse range of unanticipated worlds. The Theo-Agent architecture employs both, and uses explanation-based learning to incrementally augment its reactive component whenever forced to plan. In addition, the architecture makes widespread use of caching and dependency maintenance in order to avoid needless recomputation of repeatedly accessed beliefs. The primary characteristics of the Theo-Agent are:

- It continually reassesses what action it should perform. The agent runs in a tight loop in which it repeatedly updates its sensor inputs, chooses a control action, begins executing it, then repeats this loop.
- It reacts when it can, and plans when it must. Whenever it must choose an action, the system consults a set of stimulus-response rules which constitute its reactive component. If one of these rules applies to the current sensed inputs, then the corresponding action is taken. If no rules apply, then the planner is invoked to determine an appropriate action.
- Whenever forced to plan, it acquires a new stimulus-response rule. The new rule recommends the action which the planner has recommended, in the same situations (i.e., those world states for which the same plan justification would apply), but can be invoked much more efficiently. Learning is accomplished by an explanation-based learning algorithm (EBG (Mitchell, et al, 1986)), and provides a demand-driven incremental compilation of the planner's knowledge into an equivalent reactive strategy, guided by the agent's experiences.
- Every belief that depends on sensory input is maintained as long as its explanation remains valid. Many beliefs in the Theo-Agent, including its belief of which action to perform next, depend directly or indirectly on observed sense data. The architecture maintains a network of explanations for every belief of the agent, and deletes beliefs only when their support ceases. This caching of beliefs significantly improves the response time of the agent by eliminating recomputation of beliefs in the face of unchanging or irrelevant sensor inputs.
- It determines which goal to attend to, based on the perceived world state, a predefined set of goal activation and satisfaction conditions, and given priorities among goals.

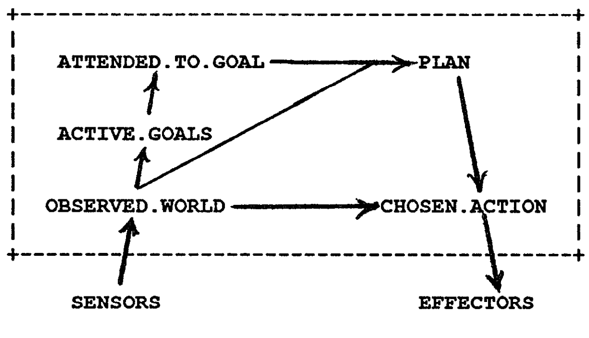

Figure 2-1: Data Flow in a Theo-Agent

Internal structure of agent: A Theo-Agent is defined by a frame structure whose slots, subslots, subsubslots, etc. define the agent's beliefs, or internal state[1]. The two most significant slots of the agent are Chosen.Action, which describes the action the agent presently chooses to perform; and Observed.World, which describes the agent's current perception of its world. As indicated in Figure 2-1 the agent may infer its Chosen.Action either directly from its Observed.World, or alternatively from its current Plan. Its Plan is in turn derived from its Observed.World and Attended.To.Goal. The Attended.To.Goal defines the goal the agent is currently attempting to achieve, and is computed as the highest priority of its Active.Goals, which are themselves inferred from the Observed.World.

Agent goals: Goals are specified to the agent by defining conditions under which they are active, satisfied, and attended to. For example, an agent may be given a goal Recharge.Battery which is defined to become active when it perceives its battery level to be less than 75%, becomes satisfied when the battery charge is 100%, and which is attended to whenever it is active and the (higher priority) goal Avoid.Oncoming.Obstacle is inactive.

Caching policy: The basic operation of the Theo-Agent is to repeatedly infer a value for its Chosen.Action slot. Each slot of the agent typically has one or more attached procedures for obtaining a value upon demand. These procedures typically access other slots, backchaining eventually to queries to slots of the Observed.World. Whenever some slot value is successfully inferred, this value is cached (stored) in the corresponding slot, along with an explanation justifying its value in terms of other slot values, which are in turn justified in terms of others, leading eventually to values of individual features in the Observed.World, which are themselves inferred by directly accessing the robot sensors. Values remain cached for as long as their explanations remain valid. Thus, the agent's Active.Goals and Chosen.Action may remain cached for many cycles, despite irrelevant changes in sensor inputs. This policy of always caching values, deleting them immediately when explanations become invalid, and lazily recomputing upon demand, assures that the agent's beliefs adapt quickly to changes in its input senses, without needless recomputation.

Control policy: The Theo-Agent is controlled by executing the following loop:

Do Forever:

1. **Sense** and update readings for all *eagerly sensed* features of Observed.World, and delete any cached values for *lazily sensed* features.
2. **Decide** upon the current Chosen.Action
3. **Execute** the Chosen.Action

When the Chosen.Action slot is accessed (during the decision portion of the above cycle), the following steps are attempted in sequence until one succeeds:

1. Retrieve the cached value of this slot (if available)
2. Infer a value based on the available stimulus-response rules
3. Select the first step of the agent's Plan (inferring a plan if necessary)
4. Select the default action (e.g., WAIT)

Sensing policy: Each primitive sensed input (e.g., an array of input sonar data) is stored in some slot of the agent's Observed.World. Higher level features such as edges in the sonar array, regions, region width, etc., are represented by values of other slots of the Observed.World, and are inferred upon demand from the lower-level features. The decision-making portions of the agent draw upon the entire range of low to high level sensory features as needed. In order to deal with a variety of sensing procedures of varying cost, the Theo-Agent distinguishes between two types of primitive sensed features: those which it *eagerly* senses, and those which it *lazily* senses. Eagerly sensed features are refreshed automatically during each cycle through the agent's main loop, so that dependent cached beliefs of the agent are retained when possible. In contrast, lazily sensed features are simply deleted during each cycle. They are recomputed only if the agent queries the corresponding slot during some subsequent cycle. This division between eagerly and lazily refreshed features provides a simple focus of attention which allows keeping the overhead of collecting new sense data during each cycle to a minimum.

Learning policy: Whenever the agent is forced to plan in order to obtain a value for its Chosen.Action, it invokes its explanation-based generalization routine to acquire a new stimulus-response rule to cover this situation. The details of this routine are described in greater detail in the next section. The effect of this learning policy is to incrementally extend the scope of the set of stimulus-response rules to fit the types of problem instances encountered by the system in its world.

3. Example and Results

This section describes the use of the Theo-Agent architecture to develop a simple program to control a Hero 2000 mobile robot to search the laboratory to locate garbage cans[2]. In particular, we illustrate how goals and actions are provided to the robot with no initial stimulus-response rules, how it initially selects actions by constructing plans, and how it incrementally accumulates stimulus-response rules that cover its routine actions.

The robot sensors used in this example include an ultrasonic sonar mounted on its hand, a rotating sonar on

[1]The Theo-Agent is implemented on top of a generic frame-based problem solving and learning system called Theo (Mitchell, et al., 1990), which provides the inference, representation, dependency maintenance, and learning mechanisms.

[2]A detailed description of the modified Hero 2000 robot used here is available in (Lin, et al., 1989).

its head, and a battery voltage sensor. By rotating its hand and head sonars it is able to obtain arrays of sonar readings that measure echo distance versus rotation angle. These raw sonar readings are interpreted (on demand) to locate edges in the sonar array, as well as regions, and properties of regions such as region width, distance, direction, and identity. The primitive sensing operations used in the present example include Battery, which indicates the battery voltage level, Sonarw, which measures sonar range with the wrist sonar pointed directly forward, and Sweep.Wrist.Roll, which obtains an array of sonar readings by rotating the wrist from left to right. Of these sensed features, Sonarw is eagerly sensed, and the others are lazily sensed.

The robot actions here include Forward.10 (move forward 10 inches), Backward.10 (move backward 10 inches), Face.The.Object (turn toward the closest sonar region in front of the robot), and Measure.The.Object (obtain several additional sonar sweeps to determine whether the closest sonar region in front of the robot is a garbage can). The.Object refers to the closest sonar region in front of the robot, as detected by the sense procedure Sweep.Wrist.Roll.

This Theo-Agent has been tested by giving it different sets of initial goals, leading it to compile out different sets of stimulus-response rules exhibiting different behaviors. In the simple example presented here, the agent is given three goals:

- Goal.Closer: approach distant objects. This goal is activated when the Sonarw sense reading is between 25 and 100 inches, indicating an object at that distance. It is satisfied when Sonarw is less that 25 inches, and attended to whenever it is active.
- Goal.Further: back off from close objects. This is activated when Sonarw is between 3 and 15 inches, satisfied when Sonarw is greater than 15 inches, and attended to whenever it is active.
- Goal.Identify.The.Object: determine whether the nearest sonar region corresponds to a garbage can. This is activated when there is an object in front of the robot whose identity is unknown, satisfied when the object identity is known, and attended to whenever it is active and Goal.Closer and Goal.Further are inactive.

In order to illustrate the operation of the Theo-Agent, consider the sequence of events that results from setting the robot loose in the lab with the above goals, actions, and sensing routines: During the first iteration through its sense-decide-execute loop, it (eagerly) senses a reading of 41.5 from Sonarw, reflecting an object at 41.5 inches. In the decide phase of this cycle it then queries its Chosen.Action slot, which has no cached value, and no associated stimulus-response rules. Thus, it is forced to plan in order to determine a value for Chosen.Action. When queried, the planner determines which goal the agent is attending to, then searches for a sequence of actions which it projects will satisfy this goal. Thus, the planner queries the Attending.To.Goal slot, which queries the Active.Goals slots, which query the Observed.World, leading eventually to determining that the Attending.To.Goal is Goal.Closer. The planner, after some search, then derives a two-step plan to execute Forward.10 two times in a row (this plan leads to a projected sonar reading of 21.5 inches, which would satisfy Goal.Closer). The inferred value for the Chosen.Action slot is thus Forward.10 (the first step of the inferred plan).

The agent caches the result of each of the above slot queries, along with an explanation that justifies each slot value in terms of the values from which it was derived. This network of explanations relates each belief (slot value) of the agent eventually to sensed features of its Observed.World.

In the above scenario the agent had to construct a plan in order to infer its Chosen.Action. Therefore, it formulates a new stimulus-response rule which will recommend this chosen action in future situations, without planning. The agent then executes the action and begins a new cycle by eagerly refreshing the Sonarw feature and deleting any other sensed features (in this case the observed Battery level, which was queried by the planner as it checked the preconditions for various actions). During this second cycle, the stimulus-response rule learned during the first cycle applies, and the agent quickly decides that the appropriate Chosen.Action in the new situation is to execute Forward.10. As it gains experience, the agent acquires additional rules and an increasing proportion of its decisions are made by invoking these stimulus-response rules rather than planning.

3.1. Rule Learning

The rule acquisition procedure used by the Theo-Agent is an explanation-based learning algorithm called EBG (Mitchell, et al, 1986). This procedure explains why the Chosen.Action of the Theo-Agent is justified, finds the weakest conditions under which this explanation holds, and then produces a rule that recommends the Chosen.Action under just these conditions. More precisely, given some Chosen.Action, ?Action, the Theo-Agent explains why ?Action satisfies the following property:

Justified.Action(?Agent, ?Action) ←
(1) The Attending.To.Goal of the ?Agent is ?G
(2) ?G is Satisfied by result of ?Agent's plan
(3) The tail of ?Agent's plan will not succeed without first executing ?Action
(4) ?Action is the first step of the ?Agent's plan

EBG constructs an explanation of why the Chosen.Action is a Justified.Action as defined above, then determines the weakest conditions on the Observed.World

```
(hero justified.action) = face.the.object
 <--prolog--
 (hero attending.to.goals) = goal.identify.object
  <--prolog--
  (hero monitored.goals) = goal.identify.object
  (hero goal.identify.object attending.to?) = t
   <--prolog--
   (hero goal.identify.object active?) = t
    <--prolog--
    (hero observed.world) = w0
    (w0 the.object identity known?) = nil
  (hero goal.closer active?) = nil
   <--prolog--
   (hero observed.world) = w0
   (w0 sonarw) = 22.5
  (hero goal.further active?) = nil
   <--prolog--
   (hero observed.world) = w0
   (w0 sonarw) = 22.5
 (world376 goal.identify.object wsatisfied?) = t
  <--prolog--
  (world376 the.object identity known?) = t
   <--expected.value--
   (world376 previous.state) = world159
   (world159 measure.the.object prec.sat?) = t
    <--prolog--
    (world159 battery) = 100
     <--expected.value--
     (world159 previous.state) = w0
     (w0 battery) = 100
      <--observed.value--
      (w0 battery observed.value) = 100
    (world159 the.object distance) = 22
     <--expected.value--
     (world159 previous.state) = w0
     (w0 face.the.object prec.sat?) = t
      <--prolog--
      (w0 battery) = 100
       <--observed.value--
       (w0 battery observed.value) = 100
      (w0 the.object direction known?) = t
     (w0 the.object distance) = 22
      <--observed.value--
      (w0 the.object distance
                     observed.value) = 22
    (world159 the.object direction) = 0
     <--expected.value--
     (world159 previous.state) = w0
     (w0 face.the.object prec.sat?) = t
      <--prolog--
      (w0 battery) = 100
       <--observed.value--
       (w0 battery observed.value) = 100
      (w0 the.object direction known?) = t
 (w0 measure.the.object prec.sat?) = nil
  <--prolog--
  (w0 the.object direction) = 10
   <--observed.value--
   (w0 the.object direction observed.value) = 10
```

Figure 3-1: Explanation for (Hero Justified.Action) = Face.The.Object

under which this explanation will hold[3]. Consider, for example, a scenario in which the Hero agent is attending to the goal Goal.Identify.The.Object, and has constructed a two-step plan: Face.The.Object, followed by Measure.The.Object. Figure 3-1 shows the explanation generated by the system for why Face.The.Object is its Justified.Action. In this figure, each line corresponds to some belief of the agent, and level of indentation reflects dependency. Each belief is written in the form (frame slot subslot subsubslot ...)=value, and arrows such as "<--observed.value--" indicate how the belief above and left of the arrow was inferred from the beliefs below and to its right. For example, the leftmost belief that the Hero's Justified.Action is Face.The.Object, is supported by the three next leftmost beliefs that (1) the (Hero Attending.To.Goals)=Goal.Identify.Object, (2) the (World376 Goal.Identify.Object Satisfied?)=t, and (3) (W0 Measure.The.Object Prec.Sat?)=nil. W0 is the current Observed.World, World376 is the world state which is predicted to result from the agent's plan, and Prec.Sat? is the predicate indicating whether the preconditions of an action are satisfied in a given world state. These three supporting beliefs correspond to the first three clauses in the above definition of Justified.Action[4]. Notice the third clause indicates that in this case the tail of the agent's plan cannot succeed since the preconditions of the second step of the plan are not satisfied in the initial observed world.

```
IF
 (1) Identity of The.Object in Observed.World
        is not Known
 (1) Sonarw in Observed.World = ?s
 (1) Not [3 < ?s < 15]
 (1) Not [25 < ?s < 100]
 (2) Battery in Observed.World > 70
 (2) Distance to The.Object in Observed.World
        = ?dist
 (2) 15 <= ?dist <= 25
 (2,3) Direction to The.Object in Observed.World
        = ?dir
 (3) Not [-5 <= ?dir <= 5]

THEN
     Chosen.Action of Hero = Face.The.Object
```

Figure 3-2: Rule for Explanation from Figure 3-1

Figure 3-2 shows the english description of the rule produced by the Theo-Agent from the explanation of Figure 3-1. The number to the left of each rule

[3]Notice that the third clause in the definition of Justified.Action requires that the first step of the plan be essential to the plan's success. Without this requirement, the definition is too weak, and can produce rules that recommend non-essential actions such as WAIT, provided they can be followed by other actions that eventually achieve the goal.

[4]The fourth clause is not even made explicit, since this is satisfied by defining the rule postcondition to recommend the current action.

precondition indicates the corresponding clause of Justified.Action which is supported by this precondition. For example, the first four lines in the rule assure that the robot is in a world state for which it should attend to the goal Goal.Identify.Object (i.e., they assure that this goal will be active, and that all higher priority goals will be inactive). Of course this rule need not explicitly mention this goal or any other, since it instead mentions the observed sense features which imply the activation of the relevant goals. Similarly, the rule need not mention the plan, since it instead mentions those conditions, labeled (2) and (3), which assure that the first step of the plan will lead eventually to achieving the desired goal.

In all, the agent typically learns from five to fifteen stimulus-response rules for this set of goals and actions, depending on its specific experiences and the sequence in which they are encountered. By adding and removing other goals and actions, other agents can be specified that will "compile out" into sets of stimulus-response rules that produce different behaviors.

3.2. Impact of Experience on Agent Reaction Time

With experience, the typical reaction time of the Theo-Agent in the above scenario drops from a few minutes to under a second, due to its acquisition of stimulus-response rules and its caching of beliefs. Let us define *reaction time* as the time required for a single iteration of the sense-decide-execute loop of the agent. Similarly, define *sensing time*, *decision time*, and *execution time* as the time required for the corresponding portions of this cycle. Decision time is reduced by two factors:

- Acquisition of stimulus-response rules. Matching a stimulus-response rule requires on the order of ten milliseconds, whereas planning typically requires several minutes.
- Caching of beliefs about future world states. The time required by planning is reduced as a result of caching all agent beliefs. In particular, the descriptions of future world states considered by the planner (e.g., "the wrist sonar reading in the world that will result from applying the action Forward.10 to the current Observed.World") are cached, and remain as beliefs of the agent even after its sensed world is updated. Some cached features of this imagined future world may become uncached each cycle as old sensed values are replaced by newer ones, but others tend to remain.

The improvement in agent reaction time is summarized in the timing data from a typical scenario, shown in table 3-1. The first line shows decision time and total reaction time for a sense-decide-execute cycle in which a plan must be created. Notice that here decision time constitutes the bulk of reaction time. The second line of this table shows the cost of producing a very similar plan on the next cycle. The speedup over the first line is due to the use of slot values which were cached during the first planning episode, and whose explanations remain valid through the second cycle. The third line shows the timing for a third cycle in which the agent applied a set of learned stimulus-response rules to determine the same action. Here, decision time (200 msec.) is comparable to sensing time (500 msec) and the time to initiate execution of the robot action (200 msec.), so that decision time no longer constitutes the bulk of overall reaction time. The decision time is found empirically to require 80 + 14r msec. to test a set of r stimulus-response rules[5].

	Decision Time	Reaction Time
1. Construct simple plan:	34.3 sec	36.8 sec
2. Construct similar plan:	5.5 sec	6.4 sec
3. Apply learned rules:	0.2 sec	0.9 sec

Table 3-1: Effect of Learning on Agent Response Time

(Timings are in CommonLisp on a Sun3 workstation)

Of course the specific timing figures above are dependent on the particular agent goals, sensors, training experience, actions, etc. Scaling to more complex agents that require hundreds or thousands of stimulus-response rules, rather than ten, is likely to require more sophisticated methods for encoding and indexing the learned stimulus-response pairings. Approaches such as Rete matching, or encoding stimulus-response pairings in some type of network (Rosenschein, 1985, Brooks, 1986) may be important for scaling to larger systems. At present, the significant result reported here is simply the existence proof that the learning mechanisms employed in the Theo-Agent are sufficient to reduce decision time by two orders of magnitude for a real robot with fairly simple goals, so that decision time ceases to dominate overall reaction time of the agent.

4. Summary, Limitations and Future Work

The key design features of the Theo-Agent are:

- A stimulus-response system combined with a planning component of broader scope but slower response time. This combination allows quick response for routine situations, plus flexibility to plan when novel situations are encountered.
- Explanation-based learning mechanism for incrementally augmenting the stimulus-response component of the system. When forced to plan, the agent formulates new stimulus-response rules that

[5]Rules are simply tested in sequence with no sophisticated indexing or parallel match algorithms.

produce precisely the same decision as the current plan, in precisely the same situations.

- The agent chooses its own goals based on the sensed world state, goal activation conditions and relative goal priorities. Goals are explicitly considered by the agent *only* when it must construct plans. As the number of learned stimulus-response rules grows, the frequency with which the agent explicitly considers its goals decreases.
- Caching and dependency maintenance for all beliefs of the agent. Every belief of the agent is cached along with an explanation that indicates those beliefs on which it depends. Whenever the agent sense inputs change, dependent beliefs which are affected are deleted, to be recomputed if and when they are subsequently queried.
- Distinction between eagerly and lazily refreshed sense features. In order to minimize the lower bound on reaction time, selected sense features are eagerly updated during each agent cycle. Other features are lazily updated by deleting them and recomputing them if and when they are required. This provides a simple focus of attention mechanism that helps minimize response time. In the future, we hope to allow the agent to dynamically control the assignment of eagerly and lazily sensed features.

There are several reasonable criticisms of the current TheoAgent architecture, which indicate its current limitations. Among these are:

- The kind of planning the TheoAgent performs may be unrealistically difficult in many situations, due to lack of knowledge about the world, the likely effects of the agent's actions, or other changes in the world. One possible response to this limitation is to add new decision-making mechanisms beyond the current planner and stimulus-response system. For example, one could imagine a decision-maker with an evaluation function over world states, which evaluates actions based on one-step lookahead (similar to that proposed in Sutton's DYNA (Sutton, 1990).). As suggested in (Kaelbling, 1986), a spectrum of multiple-decision makers could trade off response speed for correctness. However, learning mechanisms such as those used here might still compile stimulus-response rules from the decisions produced by this spectrum of decision-makers.
- Although the TheoAgent learns to become increasingly reactive, its decisions do not become increasingly correct. The acquired stimulus-response rules are only as good as the planner and action models from which they are compiled. We are interested in extending the system to allow it to inductively learn better models of the effects of its actions, as a result of its experience. Preliminary results with this kind of learning using a hand-eye robot are described in (Christiansen, et al., 1990, Zrimic and Mowforth, 1988).
- The current planner considers the correctness of its plans, but not the cost of sensing or effector commands. Therefore, the plans and the stimulus-response rules derived from them may refer to sense features which are quite expensive to obtain, and which contribute in only minor ways to successful behavior. For instance, in order to guarantee correctness of a plan to pick up a cup, it might be necessary to verify that the cup is not glued to the floor. The current system would include such a test in the stimulus-response rule that recommends the grasp operation, provided this feature was considered by the planner. We must find a way to allow the agent to choose which tests are necessary and which can be ignored in order to construct plausible plans that it can then attempt, and recover from as needed.
- Scaling issues. As noted in the previous section, the current robot system requires only a small set of stimulus-response rules to govern its behavior. We must consider how the approach can be scaled to more complex situations. Some avenues are to (1) explore other strategies for indexing learned knowledge (e.g., index rules by goal, so that many subsets of rules are stored rather than a single set), (2) develop a more selective strategy for invoking learning only when the benefits outweigh the costs, and (3) consider representations of the control function that sacrifice expressive precision for fixed computational cost (e.g., fixed topology neural networks with constant response time).

We believe the notion of incrementally compiling reactive systems from more general but slower search-based systems is an important approach toward extending the flexibility of robotic systems while still achieving respectable (asymptotic) response times. The specific design of the Theo-Agent illustrates one way to organize such a system. Our intent is to extend the current architecture by adding new learning mechanisms that will allow it to improve the correctness of its action models and its abilities to usefully perceive its world. These additional learning capabilities are intended to complement the type of learning presented here.

Acknowledgements. This work is based on extensions to earlier joint work with Jim Blythe, reported in (Blythe and Mitchell, 1989). I am most grateful for Jim's significant contributions to the design of the Theo-Agent. Thanks also to the entire Theo group, which produced the Theo system on which Theo-Agent is built. Theo provides the underlying inference, representation, and learning mechanisms used by the Theo-Agent. Finally, thanks to Long-Ji Lin who developed a number of the routines for interfacing from workstations to the robot. This research is supported by DARPA under research contract N00014-85-K-0116 and by NASA under research contract NAGW-1175.

References

[1] Agre, P. and Chapman, D.
Pengi: An Implementation of a Theory of Activity.
In *Proceedings of the National Conference on Artificial Intelligence*, pages 268-272. Morgan Kaufmann, July, 1987.

[2] Blythe, J., and Mitchell, T.
On Becoming Reactive.
In *Proceedings of the Sixth International Machine Learning Workshop*, pages 255-259. Morgan Kaufmann, June, 1989.

[3] Brooks, R.A.
A Robust Layered Control System for a Mobile Robot.
IEEE Journal of Robotics and Automation 2(1), March, 1986.

[4] Christiansen, A., Mason, M., and Mitchell, T.
Learning Reliable Manipulation Strategies without Initial Physical Models.
In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE Press, May, 1990.

[5] Kaelbling, L.P.
An Architecture for Intelligent Reactive Systems.
In M.P. Georgeff and A.L. Lansky (editor), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann , 1986.

[6] Laird, J.E. and Rosenbloom, P.S.
Integrating Planning, Execution, and Learning in Soar for External Environments.
In *Proceedings of AAAI '90*. AAAI, 1990.

[7] Lin, L., Philips, A., Mitchell, T., and Simmons, R.
A Case Study in Robot Exploration.
Robotics Institute Technical Report CMU-RI-89-001, Carnegie Mellon University, Robotics Institute, January, 1989.

[8] Mitchell, T.M., Keller, R.K., and Kedar-Cabelli, S.
Explanation-Based Generalization: A Unifying View.
Machine Learning 1(1), 1986.

[9] Mitchell, T. M., J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette, and J. Schlimmer.
Theo: A Framework for Self-improving Systems.
In VanLehn, K. (editor), *Architectures for Intelligence*. Erlbaum, 1990.

[10] Pommerleau, D.A.
ALVINN: An Autonomous Land Vehicle In a Neural Network.
In Touretzky, D. (editor), *Advances in Nerual Information Processing Systems, Vol. 1*. Morgan Kaufmann, 1989.

[11] Rosenschein, S.
Formal Theories of Knowledge in AI and Robotics.
New Generation Computing 3:345-357, 1985.

[12] Schoppers, M.J.
Universal Plans for Reactive Robots in Unpredictable Environments.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1039-1046. AAAI, August, 1987.

[13] Segre, A.M.
Machine Learning of Robot Assembly Plans.
Kluwer Academic Press, 1988.

[14] Sutton, R.
First Results with DYNA, an Integrated Architecture for Learning, Planning, and Reacting.
In *Proceedings of AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, pages 136-140. AAAI, March, 1990.

[15] Tan, M.
CSL: A Cost-Sensitive Learning System for Sensing and Grasping Objects.
In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*. IEEE, May, 1990.

[16] Zrimic, T., and Mowforth, P.
An Experiment in Generating Deep Knowledge for Robots.
In *Proceedings of the Conference on Representation and Reasoning in an Autonomous Agent*. 1988.