I Administrative

DARPA BAA 99-09:	Mobile Autonomous Robot Software
Topic Area:	Imitation (also significantly related to Soft Computing and Robot Shaping)
Title:	**Adaptive Intelligent Mobile Robots**
Technical POC:	Leslie Pack Kaelbling

II Detailed Proposal Information

A Innovative Claims for the Proposed Research

The research program proposed here is founded on three theses:

1. Neither human programming nor robot learning is alone sufficient for the construction of robust intelligent robot systems.

2. There is a continuum between reactive and deliberative behavior that represents different points in the space of trade-offs between speed and flexibility. A hierarchy of planners will operate in parallel at different points in the trade-off space.

3. Robust behavior in complex dynamic environments requires explicit representation of uncertainty.

We propose to develop, implement, and test on a mobile robot an architecture for the design of robust, flexible, robotic systems that is based on these theses. The result will be robots that can operate completely autonomously, that can accept and make use of human input in a variety of forms, and that can carry out a wide variety of tasks robustly.

The architecture will rest on a foundation of built-in reflexes and low-level behaviors that are learned with reinforcement learning methods, respecting safety constraints and taking advantage of prior knowledge supplied by human programmers. More complex behavior will arise from a hierarchy of increasingly abstract and deliberative models. It will be implemented using techniques from probability theory, Bayesian networks, neural networks and other function approximation methods, and reinforcement learning.

As a result of this research, we will have a new architecture for mobile robotics that uses both human programming and trial-and-error learning; that plans and learns hierarchically at a variety of levels of abstraction and at different points in the reactive/deliberative spectrum; and that is aware of its own knowledge and lack of knowledge and takes explicit action to gain information. This architecture will enable a revolution in mobile robot applications.

B Technical Rationale and Approach

B.1 Introduction

The theses stated above form the technical rationale for the project. In this section, we elaborate on them.

1. The necessity for learning *and* programming It is widely understood that there is no free lunch in machine learning. It is formally, technically impossible to predict a future you have never seen unless you have some information that constrains in advance the possible universes you operate in. This insight has led to the idea of *bias* in machine learning algorithms: by building in some constraints (such as the architecture of a neural network, or background knowledge given by a human, or even the encoding of a problem into a set of attributes), we can leap to appropriate conclusions about the future given knowledge of the past.

This kind of bias is crucial for robot learning, helping to constrain the kinds of hypotheses about the world that are entertained by the robot. But robots, who must learn to behave in their environments through trial and error, and who are never told explicitly what actions they should have taken, have an even more difficult problem. If they begin their "lives" without any built in knowledge, then they cannot help but choose their actions at random. And in the vast majority of environments, random actions do not cause the robot to move through the world in a way that allows them to get the crucial data that they need for learning. Imagine a robot that is supposed to learn to deliver packages. If it starts out taking random actions, it will never even make it down the hall, let alone accidentally bring me the right package so that I can reward it.

Learning robots must make use of human input in a variety of forms. They can include:

- poorly-written programs for parts of the robot's behavior
- information about the salience of some sensory patterns
- abstract advice like "When you're low on fuel go straight to the charging station"
- high-level statistical properties of the environment (like: sensory input will be encoded sufficiently redundantly that it will never be necessary to compute the parity function of more than three input bits)

A truly robust robotic learning system will have algorithms that can take advantage of human-supplied information in such a wide variety of forms. If the information is available, the system will work well. As it becomes less available or less accurate, the system will degrade gracefully.

It nearly goes without saying that direct human programming is insufficient for robust robot behavior. Humans never really understand the sensory-motor capabilities of a robot at an accurate enough level to build robust programs. Furthermore, in any kind of interesting environment, the robot will have to adapt to conditions that could not have been directly anticipated by the human.

2. Reactive and deliberative behavior: a two-dimensional continuum

The term "reactive behavior" has been defined so many times by so many people it has become nearly meaningless. Nonetheless, it seems to refer to two properties of behavior: first, that it be based entirely or largely on immediately available sensory information rather than on stored representations of historic information; and, second, that it be very quickly computed through table-lookup or a shallow-depth circuit rather than slowly via general-purpose "inference" algorithms. We argue that these two properties should be treated separately and that they are continuous rather than dichotomous.

2.1 Perception versus Memory It seems clear that, in so far as it is possible, behavior should be based on immediate sensory information. Such information can never be stale and gives the truest reflection of the current situation. However, except for the simplest behaviors, action must depend on information that is not immediately observable. This might include what was last seen when the robot looked to its left, the list of packages that need to be delivered, or the results of previous attempts to drive through some poorly-mapped territory.

The key to keeping robot behavior effective when it is based on historical information is to be very careful about representational choices. If all you need to do is avoid obstacles, it may be enough to remember a few numbers characterizing the space available on your left rather than to try to integrated it into a general-purpose 3D map of the environment. Thus, we advocate a *minimalist* approach to representation. Remember historical information to aid future behavior, but take care to remember only what you need and to take computational complexity into account when you design the representation.

2.2 Look-up versus Inference If memory were free and indexable in constant time, then the best way to make a robot would be to build a huge table, indexed by the current state of the sensory inputs, the current memory, and the current goal, containing in each cell the most appropriate action choice. Such an explicitly represented behavior or "policy" could be executed with lightning efficiency and would always do the right thing.

Sadly, memory, although it gets cheaper and bigger every day, is not free and will never be available on the scale necessary to build such a table, even for a fairly simple robot. So, in order to be able to build a robot that will react appropriately to a wide variety of stimuli, memories, and goals, we must trade off time for space. We can do this by building some general-purpose circuitry that allows us to calculate, over time, the appropriate action for the current circumstances.

This kind of time-space trade-off leads us to hierarchical architectures in which the lowest levels react quickly but relatively inflexibly, and higher levels are slower but more general purpose.

A related question comes up in the architecture of systems that learn to behave from interaction with their environment: Should they be model-based or model-free? Model-based systems first learn a model of the dynamics of their environment, then, given a goal, they can calculate (or infer) an appropriate behavior. The advantage of such systems is that the knowledge about the world transfers from one goal to the next, which is crucial in environments with changing objectives. The disadvantage is that there is an explicit inference stage between model and behavior. Model-free systems learn behavior directly with respect to a fixed goal, but previous learning is of no use when the goal changes.

In our proposed architecture, we will make use of both model-based and model-free learning strategies, often in parallel, allowing the parts of the system for which goals are fixed (self-preservation, for example) to learn a direct efficient policy, while supporting rapidly varying goals that might be supplied to a delivery or surveillance robot by a human.

3. Adaptation and Explicit Models of Uncertainty In order to operate effectively in an unknown environment, a robot must gather information. When the information to be gathered is relatively static and general purpose, we call it learning or adaptation; when the information is mostly relevant to the current situation or decision, we call it perception. In either case, information must be sought out through explicit action. The robot will try to find shortcuts on a well-traveled route; it will drive to the top of a hill for a better view of the situation; it will turn to look back to see if anyone is following or to help find its way home.

The robot must somehow know what it does and does not know, in order to take actions to explicitly improve its knowledge about the world. Such knowledge need not have esoteric representations: for learning low-level behaviors effectively, it is enough to keep some simple statistics about how many times it has tried various actions in various situations; for navigation through a complex environment, it needs to keep a probability distribution over possible places it could be rather than just imagine that it is in the most likely state.

B.2 Technical Approach

Our approach will be to build on the foundations of probability theory, Markov decision processes (MDPs), partially observable MDPs (POMDPs), reinforcement learning, neural networks and other function approximators, and Bayesian networks. These technologies give us a set of techniques for optimal planning and learning in complex uncertain environments, and we have extensive experience with all of them.

Unfortunately, but not surprisingly, the exact solution of the general versions of the problems we are interested in is computationally intractable. We cannot hope for optimal behavior in complex environments. Much of our and others' research in these areas has been in the development of approximation methods that will allow the solution of larger problems. As a result of this work, we can solve problems of modest size: robust map learning and navigation in an office building, Bayesian network inference in networks of 1000 variables, behavior learning for control of 6-dimensional robots, a world-champion Backgammon player.

The next major step is to solve *much* bigger problems. A general delivery or reconnaissance robot will live in the field for months, will traverse different kinds of terrain, will have to stop what it's doing at the moment and help with other tasks. None of the existing techniques can even begin to address a problem of this scope. In this section, we outline some of the basic existing techniques, and in the next section, we describe our planned research.

Reinforcement Learning It is abundantly clear that it is too difficult for humans to program robots directly, especially for low-level behaviors. As an alternative, we would like the robots to learn how to behave. The most mature learning technologies we have are directly *supervised*: the system is supposed to learn an input-output mapping, and is given examples of that mapping as training data. In order to apply this technique to a robot, it is necessary for a human to provide to the robot, at every decision point, an indication of the correct action. Then, the robot can learn a mapping from perceptual inputs to appropriate actions. In some special circumstances, this approach can be effective (see, particularly, Pomerleau's ALVINN system that learned to steer a car this way); however, in most cases, a human is not available or not able to provide such detailed information to the robot.

In *reinforcement learning*, the information provided by the human is much weaker. Rather than specifying what action to take at each step, the human just gives the robot an evaluation of its performance. This evaluation, called the *reinforcement signal*, may be provided directly by a human (pushing input buttons or even saying

Figure 1: Basic reinforcement-learning architecture: the robot learns to execute a mapping from sensory inputs to actions, attempting to maximize the reinforcement signal.

"good robot", "bad robot"), but is more typically provided indirectly through a program written by a human engineer. The program can automatically evaluate the robot's performance when it is out of touch with humans.

This strategy relieves the human from writing a program specifying the robot's behavior, but still requires him to write a program to evaluate the robot's performance. The human engineer still has to do a job, but that is inevitable, because we must somehow communicate to the robot what we require it to do. But the job of writing a reinforcement function is much easier than specifying a robot's behavior.

Consider the relatively simple problem of getting a robot to drive down a hallway, with sonar data as input. Specifying a robust behavior to do this is surprisingly non-trivial, as most people working in mobile robotics have discovered, to their chagrin. However, it is quite easy to write program that "rewards" the robot for making forward progress down the hall and "punishes" it for running into the walls.

A reinforcement-learning algorithm, then, has the architecture shown in figure **??**. The job of the learning algorithm is to find a mapping from inputs to actions that maximizes reinforcement.

The most naive, and easiest, strategy is to learn to take the action, on each step, that maximizes the value of the reinforcement signal on the next step. This is rarely a good idea. Consider the hallway-navigation problem described above: if the robot is rewarded for making forward progress, then its immediately best move (as long as it is oriented in roughly the right direction) is to accelerate forward as fast as possible. This is clearly a bad idea, though, because it will be impossible to stop in time to avoid collisions.

So, generally, reinforcement-learning algorithms learn to choose actions so as to maximize the long-term value. The long-term value of taking an action a in a state s, written $Q(s, a)$, is typically defined as

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a') \quad .$$

The long-term value of taking action a in state a is the immediate reinforcement for doing so, $R(s, a)$, plus the expected value of the future. We're not sure exactly what the next state will be, so we average over them, according to the probability $T(s, a, s')$ that s' will be the next state, given that we were is state s and took action a. What is the value of state s'? It's the value of the best possible action, a' that we could choose in that state: $Q(s', a')$. The factor of $0 \leq \gamma < 1$ is used to give the robot an effective planning horizon without choosing a particular fixed value; this is done by exponentially discounting the value of rewards received far into the future. The larger the value of γ, the more forward-looking the robot.

The definition of the *action-value function*, Q, given above is the formal basis for a number of learning algorithms, the best-known of which is called Q-learning (**?**). The basic strategy is to estimate the Q values based on experience. Having done so, it is easy for the robot to choose effective actions: given an input s, it just chooses the a for which $Q(s, a)$ is maximized.

A robot following this procedure would *not* choose maximum acceleration down the hallway: it would take into account the likely reinforcement values that would be gained in the future as a result of its actions, and would choose a more moderate course.

Neural Networks and Function Approximation The most basic algorithms for reinforcement learning assume that there are a small number of possible sensor states and actions, enabling the Q values to be simply represented in a table. Of course, in most realistic domains, especially robotics domains, inputs are chosen from large, typically continuous spaces.

We can deal with problems of this kind by using neural networks, or other function approximators, to represent the Q functions. During learning, rather than update the value of $Q(s, a)$ directly, we update the weights in a neural network that approximates the mapping from s to Q. This approach has been the basis of a number of successful applications, including a world-champion-level backgammon player (**?**) and an elevator scheduler (**?**). Although it has had some practical successes, it also sometimes fails to converge to a useful value function. Recent results by **?**) provide a modified training procedure that is guaranteed to converge to locally optimal performance given a particular neural-network architecture.

These strategies work very well in many cases but, as we will argue below, are probably not directly suitable for use on real robots.

Using Models So far, we have been talking about essentially reactive behavior, in the sense of both parts of thesis 2 described above: actions are chosen essentially by "lookup" rather than any kind of general deliberative process, and they depend only on the current percept, not on any historical information.

The reinforcement-learning community has begun to address the move from reactive to deliberative planning and learning in the first sense, through the use of models of world dynamics, which are formally framed as Markov decision processes (MDPs).

Basic Q learning tries to learn behavior directly, without ever learning how the world works. Another strategy is to learn a world model, which is typically some representation of $T(s, a, s')$: What is the distribution of possible next states s' given that we take action a in state s? Given a world model, there are many ways to derive behavior.

Basic Q learning is often very slow to converge, especially in domains in which actions have long-term consequences. It takes a long time for large reinforcements to propagate along chains of states; and in basic Q learning, values are only propagated when the robot takes actions in the world. An important reason for moving from "reaction" to "deliberation" is to avoid having to do too much trial and error: a robot that has learned a model of the dynamics of the world can experiment with different courses of action through mental simulation, rather than actual physical trial and error. It is better to consider choosing an action that would result in a collision and decide against it, than to collide and realize it was a bad idea.

Sutton's Dyna architecture Sutton90 and Moore and Atkeson's prioritized sweeping algorithm Moore93 provide a good start toward integrating reaction and deliberation. Each step of the agent's experience can be described by four quantities: the start state s, the action a, the resulting state s', and the reinforcement value r. Given such an experience, the agent:

- updates its estimates of the $Q(s, a)$ using the standard learning rule;
- updates its world model statistics; and
- spends the remaining computation time available before the next action using the world model to consider hypothetical actions and using its model to further update the Q values.

Algorithms of this kind use more computation power, but require much less experience in the world to learn a good strategy. Especially for robots, there is a lot of opportunity for "thought" in parallel with actions in the physical world.

So far, we have only considered domains in which there is a fixed objective, codified in the reinforcement function. And we have concentrated on finding a *policy*, which can provide an appropriate reaction to any possible state that might occur. This is appropriate for learning particular low-level motor competences. But at higher levels of abstraction, it makes little sense. We expect the objective itself to change over time (the general navigational competence should be able to be told to go to different locations, for example), and although the world is uncertain, if we know our starting state, it is easy to rule out certain resulting states as just too unlikely for consideration. Although it's possible that I will have two flat tires and a radiator leak on the way to the airport, it is just not worth the time required to plan in advance for such an eventuality.

We have done work that addresses these high-level deliberation issues, under the assumption that the agent is learning a model of the world dynamics. The Plexus system (**?**) integrated planning and reaction in stochastic domains with dynamic goals. Given a new goal, it would first generate a simple plan on the assumption that every action would have its most likely outcome. Then, given more time for deliberation, it would consider likely contingencies and determine the best courses of action for them. The longer the system had to think, the more comprehensive and robust a policy the system would generate. As the agent began to take action according to the policy, the planning would proceed in parallel, elaborating contingencies that become more likely based on the actual trajectory being taken.

In another research thread (**??**), we have pursued the automatic construction of hierarchical planning models for dynamic goals of achievement in stochastic domains. In this work, planning is done completely in advance, but the cost is much reduced due to hierarchical decomposition of the domain. This work exploits the common navigation intuition that a driver going from Chicago to New Haven can spend most of his time aiming toward New York (under the assumption that we know how to go toward New York from everywhere), and just think about New Haven when he gets into the appropriate vicinity.

Partial Observability The vast majority of the work in reinforcement learning and MDP planning assumes that the state of the world is completely observable. That is, everything that is important about the state of the world is immediately available in the current sensory signal. Of course, this assumption is false in all but the most simple-minded robotic tasks. In general, information must be integrated over time in order to determine the most appropriate action to take. This leads us to the second kind of non-reactivity: history dependence.

Although much of the theory of reinforcement learning is contingent on the satisfaction of the Markov assumption, many of the algorithms (somewhat modified) work surprisingly well in non-Markovian domains (in which, for example, many world states are perceptually indistinguishable). Still, the best these techniques can do is learn a purely reactive policy, in the sense that it does not depend on previous sensor values.

Learning history-dependent policies is a complex problem that still has no entirely satisfactory solution. Learning an exact model of the environment is hard (equivalent to HMM learning), and solving it optimally is even harder (?). However, there are many promising ideas for finding approximately optimal policies, including learning decision trees over past observations (?), searching directly in the space of policies represented as finite-state automata (?), and including explicit actions for setting and clearing bits in memory (??).

This is a crucial basic research problem which we propose to address in the course of this project.

B.3 Constructive Plan

What do we propose to do? Design, implement, and test a robot architecture with the following properties:

- Constraints on behavior that are always respected in order to maintain human safety, first, and robot safety, second. These will include basic obstacle avoidance, staying charged up, etc.
- Low-level behaviors are learned efficiently by making use of prior knowledge, especially initial badly-written programs, to bias reinforcement learning algorithms.
- A huge model of the dynamics of the world is acquired by learning small pieces at a time, represented as Bayesian networks, and then connecting them together.
- Planning is performed in an adaptive hierarchy. At the most abstract levels, it will plan for long term objectives in small models that capture only the relevant attributes of the huge model. For instance, in trying to plan to drive home from work, I might construct a model that involves local traffic and weather conditions and what we're going to have for dinner; the model does not need to involve the color of my car or the work I am going to do tomorrow. At lower, more concrete levels, the actions will be more "canned" routines that have been learned with model-free reinforcement methods and/or through caching of plans that were made frequently at higher levels.
- Information-gathering strategies are generated, also based on the current small, relevant models, that will enable effective achievement of goals.

We have extensive experience with reinforcement learning and navigation on mobile robots, in addition to expertise in probabilistic models of inference and planning, supervised learning, and reinforcement learning. The following sections lay out concrete technical proposals for achieving these goals.

In the first two years, the project would have two parallel efforts.

The first would be to really demonstrate the utility of reinforcement learning (RL) for robots. We will consider RL to be a success when we can get a robust desired behavior with an amount of effort (including both human effort to supply prior knowledge and robot effort to learn) that is less than it would have taken for the human to program the robot directly. There have been no compelling demonstrations of this, but we are convinced that we can make this happen with good exploitation of initial approximate programs, as described in the first section below. Once this basic demonstration was done, we would concentrate on learning behaviors that respect given safety constraints and that incorporate other kinds of human advice, in the context of the second section below.

The second effort would be to invent the more complex architectural features described in the last two sections. We understand, in broad terms, how to do this, but there is a great deal of invention and innovation required to get it right. During the invention phase, these features would be implemented and tested in simulated robotic models.

The second two years would also consist of two parallel efforts. In the first effort, we would transition the RL technology to another robotic platform and to a wide variety of tasks. We would also expand the forms of advice and knowledge that humans could give to the robots. In the second effort, we would implement the entire integrated architecture for hierarchical planning and learning on our robot, and give a demonstration of long-term, complex, intelligent behavior.

Making reinforcement learning really work for robots Reinforcement-learning techniques have been successful in a number of large applications; but in every case, they have either had extremely strong *a priori* constraints put on the policy or have been learned primarily in simulation with *huge* amounts of experience. Our first goal is to make RL work *online* on a real robot. We are interested in domains with continuous sensory inputs and continuous actions, and require that learning take place online from a relatively small amount of experience.

Our first step will be to use nearest neighbor (and locally weighted regression) as a function approximator for the Q functions, rather than the more traditional choice of sigmoidal neural networks. There are three important reasons for this:

- Nearest neighbor learns from a single presentation of a data point; there is no requirement for multiple presentations of the same data.

- Nearest neighbor learning is local: when the sampling distribution changes over time (as it inevitably does when gathering real experience from a robot), there is no catastrophic forgetting, as there is in typical neural network models.

- It is possible, with nearest neighbor, to estimate the validity of predictions based on the density of experience in the area; this can keep Q-learning from going out of control due to bad extrapolations.

Preliminary experiments show that this change in function approximators is likely to make online RL much more effective.

There are still a number of technical issues to be dealt with:

- Nearest neighbor requires a distance metric on the input/output space, which may need tuning for best performance.

- Traditional Q learning requires a maximization over a finite set of actions. We have an infinite action space and cannot find the true maximum; we are developing strategies for choosing better actions based on a relatively small number of probes of current Q values.

- Indexing and editing schemes will have to be employed to keep memory from being overrun.

- Q learning requires a function approximator that can track a changing function; this means that older points will have to have less weight in nearest-neighbor predictions.

We have concrete strategies for addressing each of these problems. They are performing well in small pilot applications, but still need development and tuning for a larger application.

As a result of this task, we will have concrete proof that suitably modified versions of current reinforcement-learning techniques can perform well on real mobile robots.

An architecture for integrating programming and learning We expect the methods of the previous section to work well for learning low-level tasks in which actions do not have extremely long-term consequences. As we consider more sophisticated tasks, with longer time horizons, it will surely be the case that Q learning will not be effective in a reasonable amount of time. There are two main problems (both mentioned above):

- Undirected exploration will lead to a random walk, which can take unacceptably long to discover interesting parts of the space.

Figure 2: Learning architecture: samples from T are used to train Q; maximal values of $Q(s, a)$ are used to train π.

- Model-free learning cannot benefit from hypothetical consideration of actions and requires too much trial-and-error experience in the real world.

Our second task will be to develop, implement, and test an architecture for reinforcement learning that addresses both of these difficulties. The block structure is shown in figure **??**; it will work as follows:

- The user provides a policy, π_u, for performing the desired behavior. This policy can be very bad; its crucial role is to bias the robot's initial behavior enough so that the early actions of the robot will lead to interesting experience, giving bootstrapping help to the learning algorithms. This policy is a black box from the perspective of this architecture; it can be a program in any language, or even a human with a joystick.

- In the first phase of the learning, the robot uses supervised learning techniques to train its internal policy π to choose the same actions as π_u, on the grounds that these are the best action choices we have now. Simultaneously, it performs the Q-learning algorithm above to begin estimating the action-value function, *and* uses its experience to train a world model, learning to predict the next state distribution as a function of the current state and action.

- As the world model becomes more accurate, it will be used to generate hypothetical experience, as in Dyna, for training the Q functions.

- As the Q functions become more accurate, they will be used to choose actions, and those action choices will be used to train π, overwriting and improving the originally provided policy.

Figure 3: A variable-resolution view of the environment, in which closer, more-likely situations are considered in detail, and future, and less-likely situations are considered abstractly.

We believe that the use of an initial human-provided policy will allow the robot to get much more useful experience much more quickly, and that such bad-but-useful policies will not be hard to provide. Furthermore, the deliberative use of a model will allow much more efficient "compilation" of experience into good reactions.

Additionally, it is quite easy to incorporate low-level "safety" constraints on the robot's actions. Given a black-box procedure that maps an input situation into a legal (or illegal) range of actions, the Q-learning action-choice mechanism can always maximize subject to the constraints on allowable actions. This is an additional benefit of using Q functions to represent our knowledge about the desirability of actions in the domain: it is much easier to choose the best action subject to a constraint than it would be if all we had was a direct representation of the policy.

A hierarchy of deliberative levels We understand the previous two tasks well enough to begin implementing and testing them directly on a robot. This section and the next describe much more basic research; we are confident that we can arrive at good solutions, but are not yet in a position to characterize them completely. The tasks are aimed at increasing the scope of the tasks addressable by the robot system by expanding the role of deliberation.

We propose to construct a hierarchical version of the Plexus system (described above) for "deliberative" planning in response to dynamic goals. At all times, the robot will have plans at multiple levels of abstraction. The higher the level of abstraction, the longer the horizon of the plan; the lower the level of abstraction, the more detailed the plan.

Another way to think of this is to imagine the robot with a variable-resolution view of its environment, as shown in figure **??**. Parts of the world that are "close" to the robot, because they are likely to be encountered in the near future are considered at a very fine grain. Other parts of the world, that are either very unlikely to occur or that will be relevant in the future are considered much more abstractly. Thus, the robot might simultaneously

- choose its current motor actions based on a very fine view of the obstacles immediately present;
- decide whether to go to the charging station based on a more abstract view of the amount of fuel it is likely to use on the remainder of this surveillance mission; and
- decide which surveillance mission to do next based on much more abstract assessment of the enemy's likely motion.

There are two motivations for a design of this sort. First, in many cases, there just isn't enough information available to plan future courses of action in detail. There is no point in thinking about what path you will take across a room that you have never encountered before; you just have to believe that you can find such a path when you get there, and have a rough estimate of how long it's likely to take. Even if complete information were available, however, complete planning at the most detailed level is almost inevitably intractable. It is practical to give up a certain amount of reinforcement in exchange for a huge computational savings.

In order to represent states at variable levels of resolution, it will be necessary to move to factored representations of states, in terms of state variables. Then, the state-transition distribution can be represented compactly by a dynamic Bayesian network.

The Plexus system could be thought of as an extremely simple instance of this architecture: states are either considered in complete detail, or are lumped into one large abstract state. Given our experience with this system, as well as with the automatic generation of and planning in hierarchical models, and with learning Bayesian network representations, we believe we can design an effective architecture along these lines.

Partial observability So far, we have left out entirely the problem of memory. In any kind of an interesting domain, perception will not reveal everything that is important about a current state of the world. It will be important to remember information that was learned on previous time steps, while being careful to remember that it may degrade over time in a dynamic domain (if I look over my shoulder to see a car in the next lane while I'm driving, then as time passes, I'm less and less sure about that car's position relative to mine).

Interestingly enough, what's hard is not remembering things from the past; what's hard is forgetting! In order to keep from being hopelessly clogged up with useless information, we must remember only those things that will be useful to us in the future. We have two strategies in mind for this: one particular and one more general.

In the process of learning the world model, we might find that there's nothing in the current state that will let us predict reliably what we'll see when we open a box. We wish to find a strongly predictive model, so we are driven to consider other explanations. One explanation is that some information we might have had in the past will let us predict what's in the box. So, we begin to search for past information that will improve our predictions. Once we realize that it's important to remember what's in the box, we will do so as a matter of policy in the future.

Another strategy is to learn general salience properties in the environment. It may be that the locations of nearby fuel stations turn out to be important for many different predictions. If we see this pattern occurring, we may decide that fuel stations are particularly salient and make a policy of remembering their locations, even without an obvious immediate motivation to do so. We expect that this kind of salience learning or "learning what is important" will be crucial for survival in huge domains. It is a way to take a huge problem and immediately abstract it into the important attributes.

If something really surprisingly awful or wonderful happens, that is another cue for salience. We may notice particular attributes of such states (or the situations leading up to such states) and pay careful attention to them in the future.

Most existing solutions to learning in POMDPs couple closely learning control strategies with learning what to remember. This approach is very well motivated, but it may ultimately be too computationally intractable. The strategies described above, together with human guidance as to what features in an environment are important to attend to and remember, may allow us to decouple the problem of learning what to remember from the problem of learning how to behave given what we have remembered. We believe that human guidance of this kind should be quite easy to provide, and therefore propose to pursue a decoupled solution to the POMDP problem.

In addition to deciding what to remember, a truly intelligent robot will have to take actions for the explicit purpose of gaining information. The POMDP framework enables this kind of action without any special effort. If we consider the "state" in our control problem to be a memory state of the robot, rather than a state of the world, then the goal is to learn to choose actions based on the current memory state in order to drive the system into good memory states. Thus, when the robot is lost (its memory state does not indicate reliably where it is), it might be most appropriate to drive to the top of the hill, or to ask another robot for information. All actions are viewed as changing the memory state of the robot as well as changing the external state of the world, and are evaluated in light of both of these effects.

We have a great deal of experience in the exact and approximate solution of POMDPs and expect to be able to bring it to bear effectively on this problem.

B.4 Conclusions

As a result of this research, we will have a new architecture for mobile robotics that uses both human programming and trial-and-error learning; that plans and learns hierarchically at a variety of levels of abstraction and at different points in the reactive/deliberative spectrum; and that is aware of its own knowledge and lack of knowledge and takes explicit action to gain information. This architecture will enable a revolution in mobile robot applications.

We conclude with a brief reminder of how this proposal addresses the required capabilities.

A software framework for hybrid deliberative/reactive control Basic reinforcement learning is entirely reactive, by anyone's definition. We introduce deliberation through the use of a world model to allow learning from hypothetical experience, the use of a hierarchy of planners to allow flexible, multi-resolution domain models, and the use of POMDP models to allow dependence of actions on the history of percepts.

A software composition methodology Learning and programming are integrated by allowing humans to supply information and constraint at a variety of levels. We have mentioned the following in particular, but it would be possible to integrate a variety of other kinds of pre-specified information:

- initial bad policies to bias exploration;
- constraints on action choices for reasons of safety; and
- information about what attributes of the environment are important to aid feature selection and memory strategies.

Safe operation in close proximity to humans It is easy to integrate world knowledge, represented as Q functions, with inviolable safety constraints on action choices. However, if safe operation is to be guaranteed, the constraints will have to be hand-programmed in advance; it is, of course, nonsensical to imagine learning such constraints through experimentation.

Prototype software demonstrations and experiments The first two tasks will be demonstrated on a real mobile robot that learns to drive through the hallways in an office environment. Experiments and demonstrations will take place with a variety of sensory input, including sonar, laser range-finder data, and vision (predominantly optical flow, with which we have had success at building robust navigation systems). The robot will learn to perform low-level tasks such as driving down hallways, going through doorways, and visually servoing to landmark locations quickly and robustly. When coupled with some global positioning information (GPS, compass, path integration), the robot will be able to learn to compose these behaviors to go to particular locations.

We will have succeeded in this project if our methodology for combining learning with programming results in quicker construction of more robust systems than direct hand-coding would have.

The second two tasks will be addressed in simulation during the first phase of the project. The solutions developed during that phase will be migrated to the real robot during the optional years. The tasks to be addressed here will be much more complex, involving multiple delivery and surveillance tasks and long life-times. Tasks of this magnitude have not been successfully addressed by hand-coding or any other method; any ability to solve them will be considered a success of this part of the project.

C Deliverables

We shall make no proprietary claim to the results of the proposed research.

The deliverables of this project consist of technical papers describing the results of this research. Such results will include: new algorithms for robot learning and planning, as well as experimental data from testing the algorithms in the test domains.

At the end of each funding year, a demonstration of prototype software will be made available for the sponsors.

D Statement of Work

We will exert our best efforts to perform the following tasks. All progress will be documented in technical papers.

Base effort

- Implement extended Q-learning in continuous state and action spaces, based on nearest-neighbor generalization as described in the body of the proposal.
- Test and demonstrate extended Q learning on hallway tasks for real robot.
- Refine architecture for integrated planning and learning, deliberation and reaction; develop associated algorithms.
- Implement integrated architecture.
- Test and demonstrate integrated architecture on real robot.
- Develop algorithms for hierarchical deliberative planning.
- Test algorithms for hierarchical deliberative planning in simulated domain.
- Develop algorithms for efficient learning and planning in large partially observable domains.
- Invent algorithms and data structures that allow the representation of the state of the world, at least in part, in terms of objects and their properties and relations. Find algorithms for learning world models in this representation and for doing efficient planning.

Option 1

- Migrate extended Q-learning and integrated architecture demonstration to sponsor-provided robot platform.
- Implement hierarchical deliberative planning on real robot; integrate with existing robot architecture.
- Test and demonstrate hierarchical deliberative planning on real robot.
- Test algorithms for efficient learning and planning in simulated large partially observable domains.
- Test object-based representations in large simulated domain.

Option 2

- Demonstrate application of hierarchical deliberative planning to an additional large mobile robot task.
- Implement algorithms for efficient learning and planning for partially observable domains on real robot.
- Implement object-based representation, learning, and planning on real robot.
- Test and demonstrate integration of all phases of research on large, sustained domain of delivery and surveillance on real robot.

E Schedule of Milestones

It is impossible to schedule research breakthroughs with high precision. We therefore present this schedule on a yearly basis.

Year 1

- Implement extended Q-learning in continuous state and action spaces, based on nearest-neighbor generalization as described in the body of the proposal.
- Test and demonstrate extended Q learning on hallway tasks for real robot.
- Refine architecture for integrated planning and learning, deliberation and reaction; develop associated algorithms.
- Develop algorithms for hierarchical deliberative planning.

Year 2

- Implement integrated architecture.
- Test and demonstrate integrated architecture on real robot.
- Test algorithms for hierarchical deliberative planning in simulated domain.
- Develop algorithms for efficient learning and planning in large partially observable domains.

Year 3 Tasks in Option 1, as described in section D.

Year 4 Tasks in Option 2, as described in section D.

References

Minoru Asada, Takayuki Nakamura, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23, 1996.

Leemon C. Baird and Andrew W. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. The MIT Press, 1999.

Craig Boutilier, Thomas Dean, and Steve Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, to appear.

Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 9*. The MIT Press, 1997.

Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76, 1995.

Marco Dorigo and M. Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71, 1994.

Jeff Forbes, Tim Huang, Keiji Kanazawa, and Stuart Russell. The BATmobile: Towards a bayesian automated taxi. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

Milos Hauskrecht, Nicolas Meuleau, Craig Boutilier, Leslie Pack Kaelbling, and Thomas Dean. Hierarchical solution of Markov decision processes using macroactions. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, Madison, Wisconsin, 1998.

Leslie Pack Kaelbling. Goals as parallel program specifications. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis-St. Paul, Minnesota, 1988.

Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts, 1993a. Morgan Kaufmann.

Leslie Pack Kaelbling. *Learning in Embedded Systems*. The MIT Press, Cambridge, Massachusetts, 1993b. Also available as a PhD Thesis from Stanford University, 1990.

Leslie Pack Kaelbling, Anthony R. Cassandra, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Osaka, Japan, 1996a.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996b.

Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In *From Animals to Animats 3*, Brighton, UK, 1994.

Mario Martin. *Reinforcement Learning for Embedded Agents facing Complex Tasks*. PhD thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 1998.

Maja J. Mataric. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4, 1997.

Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, New York, 1995.

Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled Markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, 1998. AAAI Press.

Nicolas Meuleau, Kee-Eung Kim, Milos Hauskrecht, Anthony R. Cassandra, and Leslie Pack Kaelbling. Solving POMDPs by searching the space of finite policies. (manuscript, submitted to IJCAI99), 1999.

Andrew Moore, Leemon Baird, and Leslie Pack Kaelbling. Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. (manuscript, submitted to IJCAI99), 1999.

Andrew W. Moore and Christopher G. Atkeson. Memory-based reinforcement learning: Efficient computation with prioritized sweeping. In *Advances in Neural Information Processing 5*, San Mateo, California, 1993. Morgan Kaufmann.

Stanley J. Rosenschein and Leslie Pack Kaelbling. A situated view of representation and control. *Artificial Intelligence*, 72(2), 1995.

Stefan Schaal. Learning from demonstration. In *Advances in Neural Information Processing Systems 9*. The MIT Press, 1997.

Stefan Schaal and Christopher Atkeson. Robot juggling: An implementation of memory-based learning. *Control Systems Magazine*, 14, 1994.

Jeff Schneider, Justin Boyan, and Andrew Moore. Value function based production scheduling. In *Proceedings of the International Conference on Machine Learning*, Madison, Wisconsin, 1998.

Hagit Shatkay and Leslie Pack Kaelbling. Learning topological maps from weak odometric information. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.

Hagit Shatkay and Leslie Pack Kaelbling. Heading in the right direction. In *Proceedings of the Fifteenth International Conference on Machine Learning*, Madison, Wisconsin, 1998.

Peter Stone and Manuela Veloso. Team-partitioned, opaque-transition reinforcement learning. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, Washington, 1999.

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, Texas, 1990. Morgan Kaufmann.

Richard S. Sutton, Doina Precup, and Satinder Singh. Intra-option learning about temporally abstract actions. In *Proceedings of the Fifteenth International Conference on Machine Learning*, Madison, Wisconsin, 1998.

Gerald J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8, 1992.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.