High-level robot behavior control using POMDPs

Joelle Pineau and Sebastian Thrun
Robotics Institute and Computer Science Department
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh PA 15213
{jpineau,thrun}@cs.cmu.edu

Abstract

This paper describes a robot controller which uses probabilistic decision-making techniques at the highest-level of behavior control. The POMDP-based robot controller has the ability to incorporate noisy and partial sensor information, and can arbitrate between information gathering and performance-related actions. The complexity of the robot control domain requires a POMDP model that is beyond the capability of current exact POMDP solvers, therefore we present a hierarchical variant of the POMDP model which exploits structure in the problem domain to accelerate planning. This POMDP controller is implemented and tested onboard a mobile robot in the context of an interactive service task. During the course of experiments conducted in an assisted living facility, the robot successfully demonstrated that it could autonomously provide guidance and information to elderly residents with mild physical and cognitive disabilities.

Introduction

High-level robot control has been a popular topic in AI, and decades of research has led to a reputable collection of architectures (e.g., (Arkin 1998; Brooks 1985; Gat 1996)). However, existing architectures rarely take uncertainty into account during planning. In this paper we describe a high-level robot control system that uses probabilistic decision-making to act under uncertainty.

Partially Observable Decision Processes (POMDPs) (Sondik 1971) are techniques for calculating optimal control actions under uncertainty. They extend the well-known Markov Decision Processes (MDPs) (Howard 1960) to domains where considerations of noise and state uncertainty are crucial to good performance. They are useful for a wide range of real-world domains where joint planning and tracking is necessary, and have been successfully applied to problems of robot navigation (AAAI 1998; Simmons & Koenig 1995; Nourbakhsh, Powers, & Birchfield 1995; Roy & Thrun 2000) and robot interaction (Darrell & Pentland 1996; Roy, Pineau, & Thrun 2000).

In this paper we describe a system that uses POMDPs at the highest-level of behavior control, in contrast with existing POMDP applications in robotics where POMDP control is limited to specialized modules. We propose a robot control architecture where a POMDP performs high-level control by arbitrating between information gathering and performance-related actions, as well as negotiating over goals from different specialized modules. The POMDP also incorporates high-level uncertainty obtained through both navigation sensors (e.g. laser range-finder) and interaction sensors (e.g. speech recognition and touchscreen.)

Unfortunately, POMDPs of the size necessary for good robot control are an order of magnitude larger than today's best exact POMDP algorithms can tackle (Kaelbling, Littman, & Cassandra 1998). However, the robot controller domain yields a highly structured POMDP, where certain actions are only applicable in certain situations. To exploit this structure, we developed a *hierarchical* version of POMDPs, which breaks down the decision making problem into a collection of smaller problems that can be solved more efficiently. Our approach is similar to the MAX-Q decomposition for MDPs (Dietterich 2000), but defined over POMDPs (where states are unobserved.)

Finally, we apply the high-level POMDP robot controller to the real-world task of guiding elderly people in a nursing home. In systematic experiments, the robot successfully demonstrated that it could autonomously provide guidance for elderly residents and we found the POMDP approach to be highly effective.

Review of POMDPs

This section provides a brief overview of the essential concepts in POMDPs (see (Kaelbling, Littman, & Cassandra 1998) for a more complete description of the POMDP problem formulation.)

A POMDP model is an n-tuple, $P = \{S, A, \Omega, b_0, T, O, R\}$, consisting of:

- *States*: A set of states, $S = \{s_1, s_2, ...\}$, describes the problem domain. The domain is assumed to be in a specific state s_t at any point in time.
- *Actions*: A set of actions, $A = \{a_1, a_2, ...\}$, describes the agent's interaction with the domain. At any point in time the agent applies an action, a_t, through which it affects the domain.
- *Observations*: A set of observations, $\Omega = \{o_1, o_2, ...\}$, describes the agent's perception of the domain. A received observation, o_t, may only partially reflect the current state.
- *Rewards*: A set of numerical costs/rewards, $R(s_t, a_t)$, describes the reinforcement received by the agent throughout its interaction with the domain.

To fully characterize a specific POMDP model, the following probability distributions must be specified:

- *Initial state probability distribution*:

$$b_0(s) := Pr(s_0 = s) \quad (1)$$

  is the probability that the domain is in state s at time $t = 0$. This distribution is defined over all states in S.

- *State transition probability distribution*:
$$T(s,a,s') := Pr(s_t = s' | s_{t-1} = s, a_{t-1} = a) \quad (2)$$
is the probability of transitioning to state s', given that the agent is in state s and selects action a. This is defined for all (s,a,s') triplets and assumes $\sum_{s' \in S} T(s,a,s') = 1$, $\forall (s,a)$.
- *Observation probability distribution*:
$$O(s,a,o) := Pr(o_t = o | s_{t-1} = s, a_{t-1} = a) \quad (3)$$
is the probability that the agent will perceive observation o, given that it is in state s, and has applied action a. This is defined for all (s,a,o) triplets and assumes $\sum_{o \in S} O(s,a,o) = 1, \forall (s,a)$.

At any given point in time, the system is assumed to be in some state s_t, which may not be completely observable, but is partially observable through observation o_t. In general, it is not possible to determine the current state with complete certainty. Instead, a *belief* distribution is maintained to succinctly represent the history of the agent's interaction (both applied and perceived) with the domain:

- *Belief*:
$$b_t(s) := Pr(s_t = s | o_t, a_{t-1}, o_{t-1}, ..., a_0, b_0) \quad (4)$$
describes the probability that, at time t, the agent is in state s_t, given the history $\{o_t, a_{t-1}, o_{t-1}, ..., a_0\}$ and the initial belief b_0. This distribution is defined over all states in S.

Assuming a POMDP model as defined above, we now discuss two interesting problems in POMDPs. The first is state tracking, that is, the computation of the belief state at each time step. The second is the optimization of an action selection policy.

State Tracking

To operate in its domain and apply a belief-conditioned policy, an agent must constantly update its belief distribution. Equation 5 defines the update rule for computing a posterior belief, b', given the belief at the previous time step, b, and the latest action/observation pair, (a,o):

$$b'(s') = \frac{O(s',a,o) \sum_{s \in S} T(s,a,s') b(s)}{Pr(o|a,b)} \quad (5)$$

For most POMDP domains, state tracking is easy, relative to the problem of computing a useful action selection policy. For very large scale problems the belief updating may become problematic, this has been addressed by earlier work, in the context of dynamic Bayesian networks (Boyen & Koller 1998; Poupart & Boutilier 2000) and probabilistic tracking (Thrun *et al.* 2000b).

Computing a Policy

A POMDP policy, π, is an action-selection strategy. It is formally defined as a mapping from *belief* to action selection:

$$\pi^{POMDP} : b_t \rightarrow a \quad (6)$$

In this, POMDPs differ significantly from MDPs, where the policy is a mapping from *state* to action:

$$\pi^{MDP} : s_t \rightarrow a \quad (7)$$

The goal of planning, or POMDP problem solving, is to learn an action-selection policy that maximizes the (possibly discounted) sum of expected future reward up to some time T:

$$E[\sum_{\tau=t}^{T} R(\tau)] \quad (8)$$

where $R(\tau)$ is the reward at time τ.

The most straight-forward approach to finding POMDP policies remains the value iteration approach, where iterations of dynamic programming are applied to compute increasingly more accurate values for each belief state b.

$$V : B \rightarrow R \quad (9)$$

After convergence, the value is the expected sum of all (possibly discounted) future pay-offs the agent expects to receive up to time T, if the correct belief is b. A remarkable result by Sondik (Sondik 1971) shows that for a finite-horizon problem, the value function is a piecewise linear, convex, and continuous function of the belief, with finitely many linear pieces.

Unfortunately however, the exact algorithms used to compute optimal policies are bounded by a doubly exponential computational growth in the planning horizon, and in practice are often at least exponential. More specifically, a single step of value iteration is on the order of

$$|\Gamma_t| = O(|A||\Gamma_{t-1}|^{|\Omega|}) \quad (10)$$

where $|\Gamma_{t-1}|$ represents the number of components necessary to represent the value function at iteration $t-1$. This points to the need for more efficient algorithms.

Recent efforts have focused on the development of efficient algorithms that use value-approximation techniques to generate near-optimal policies for large domains (Hauskrecht 2000; Littman, Cassandra, & Kaelbling 1995). However many of these algorithms, though scalable and highly effective in many domains, generally gain computational advantage at the expense of information-gathering considerations, thus making them inappropriate for domains where information-gains are crucial to good performance.

An alternative approach for scaling decision-making is to exploit hierarchical structure and partition a complex problem into many smaller problems. The idea of leveraging structure has been extensively studied in the context of Markov Decision Processes (MDPs) (Singh 1992; Dayan & Hinton 1993; Kaelbling 1993; Dean & Lin 1995; McGovern *et al.* 1998; Parr & Russell 1998; Dietterich 2000). These algorithms do not extend naturally to POMDPs because they define structured hierarchies in terms of modular subtasks with *fully observable* start and termination conditions. We now describe a novel algorithm to perform hierarchical POMDP planning and execution.

Hierarchical POMDPs

The basic idea of the hierarchical POMDP is to partition the action space—not the state space, since the state is not fully observable—into smaller chunks. Therefore the cornerstone of our hierarchical algorithm is an *action hierarchy*. We assume that it is provided by a designer and represents the structural prior knowledge included to facilitate the problem solution. Figure 1 illustrates the basic concept of an action hierarchy.

Formally, an action hierarchy is a tree, where each leaf is labeled by an action from the target POMDP problem's action

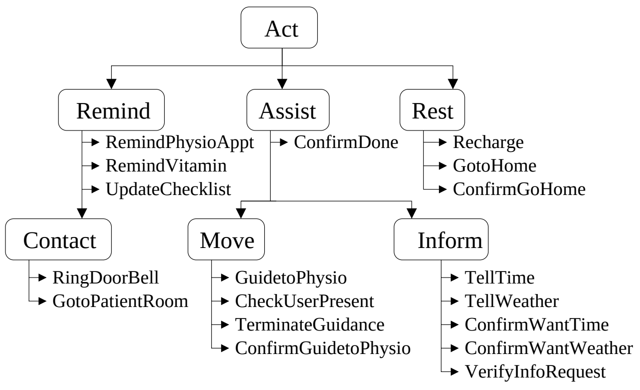

Figure 1: Sample Action Hierarchy

set. Each action $a \in A_0$ (henceforth called *primitive action*) must be attached to at least one leaf (e.g. *RingDoorBell*, *GotoPatientRoom*, etc.) In each internal node (shown as circles in figure 1) we introduce an *abstract action*. Each of these provides an abstraction of the actions in the nodes directly below it in the hierarchy (e.g. *Contact* is an abstraction of *RingDoorBell* and *GotoPatientRoom*.) Throughout this paper we use notation $\bar{a}_i$ to denote an abstract action.

A key step towards hierarchical problem solving is to use the action hierarchy to translate the original full POMDP task into a collection of smaller POMDPs. The goal is to achieve a collection of POMDPs that individually are smaller than the original POMDP, yet collectively define a complete policy.

We propose that each internal node $\bar{a}_i$ in the action hierarchy spans a corresponding subtask $\mathbf{P}_i$. The subtask is a well-defined POMDP composed of:

- a state space S_i, identical to the full original state space S_0;
- an observation space Ω_i, identical to the full original observation space Ω_0;
- an action space A_i, containing the children nodes (both *primitive* and *abstract*) immediately below $\bar{a}_i$ in the subtask.

For example, figure 1 shows a problem that has been divided into seven subtasks, where subtask P_{remind} has action set: A_{remind}=$\{a_{RemindPhysioAppt}, a_{RemindVitamin}, a_{UpdateChecklist}, \bar{a}_{Contact}\}$, and so on.

Given that the action hierarchy spans a collection of separate POMDP subtasks, we can independently optimize an independent policy for each subtask, such that we obtain a collection of corresponding *local policies*:

DEFINITION: *Given $\mathbf{P}_i$, a POMDP subtask with action set A_i. We say that π_{P_i}, a POMDP policy defined over action subset A_i, is a* local policy.

Using this representation, we introduce two key algorithms: a hierarchical planning algorithm which is used to optimize local policies for all subtasks, and a hierarchical execution algorithm which is used to extract a global action policy from the collection of local policies.

Hierarchical POMDP Execution

We first present the execution algorithm, which assumes that local policies have been computed for all subtasks and from these extracts a global policy mapping belief to action. Rather than computing a full global policy, we propose an online algorithm that consults the local policies at every time step. Before doing a the policy lookup, the belief is first updated using equation 5 (except at time $t = 0$, where the initial belief $b_t = b_0$). The execution algorithm then begins by invoking the policy of the top subtask in the hierarchy, and then operates through a sequence of recursive calls such that the hierarchy is traversed in a *top-down* manner, invoking a sequence of local policies until a primitive action is selected. The precise procedure is described in the EXECUTE function of table 1. The routine is initially invoked using the top subtask as the first argument, P, and the current belief as the second argument, b_t.

```
EXECUTE($P$, $b_t$)
        Let $a_i = \pi_P(b_t)$
        If $a_i$ is a primitive action
                Return $a_i$
        Else if $a_i$ is an abstract action (i.e. $\bar{a}_i$
                Let $P_{child}$ be the subtask spanned by $\bar{a}_i$
                EXECUTE($P_{child}$, $b_t$)
        end
end
```

Table 1: Execution function

It is important to emphasize that the full top-down trace through the hierarchy is repeated at every time step. This is a departure from many hierarchical MDP planning algorithms which operate within a given subtask for multiple time steps until a terminal state is reached. It is however consistent with Dietterich's polling approach (Dietterich 2000), and in our approach is strongly motivated by the fact that the partial observability characteristic of POMDPs would limit the detection of terminal states, thus preventing us from using the execution approach common to hierarchical MDPs.

Hierarchical POMDP Planning

We now describe the planning algorithm which is responsible for optimizing the collection of local policies. Table 2 describes in pseudo-code the planning algorithm.

```
PLAN($P$)
        forall primitive actions $a_j$ in $P$
                PARAMETERIZE($a_j$)
        end
        forall abstract actions $\bar{a}_m$ in $P$
                Let $P_m$ be the subtask spanned by $\bar{a}_m$
                PLAN($P_m$)
                PARAMETERIZE($\bar{a}_m$)
        end
        SOLVE($P$)
end
```

Table 2: Planning function

The recursive PLAN(P) routine is responsible for traversing the hierarchy and is initially invoked using the top subtask in the hierarchy as the argument. However it traverses the hierarchy in a *bottom-up* manner. Namely, each subtask is solved only after all the subtasks directly below it in the hierarchy have been solved. The routine uses a simple two-part process, applied in succession to all subtasks in the hierarchy.

- *PART 1* (PARAMETERIZE): *Infer the necessary parameters for the given subtask.*
- *PART 2* (SOLVE): *Apply policy optimization to the subtask.*

The SOLVE function contains the actual policy optimization subroutine, which is implemented as a call to an exact POMDP planning algorithm: the incremental pruning algorithm. This exact POMDP solution is described in detail in (Cassandra, Littman, & Zhang 1997); implemented code can be obtained from (Cassandra 1999). The assumption is that each subtask is sufficiently small to be solved exactly, yet the full POMDP problem is much too large for an exact solution.

The PARAMETERIZE routine is used to infer the necessary model parameters (transition, observation, and reward) for each subtask. We assume we have a known model of the original full POMDP task, and use these to infer a subset of parameters for each subtask. We consider two separate issues: that of defining parameters for *primitive actions* (i.e. PARAMETERIZE(a_j)), and that of defining parameters for *abstract actions* (i.e. PARAMETERIZE($\bar{a}_m$).) In general, the parameters that are conditioned on primitive actions can be directly extracted from the original parameter set. We formally define:

Initial belief (11)

$$b_i(s) := b_0(s), \;\; \forall s \in S_0$$

Transition probabilities (12)

$$T_i(s, a_j, s') := T_0(s, a_j, s'), \;\; \forall (s, s') \in S_0, \;\; \forall \{a_j, ..., a_l\} \in A_i$$

Observation probabilities (13)

$$O_i(s, a_j, o) := O_0(s, a_j, o), \;\; \forall s \in S_0, \;\; \forall o \in \Omega_0, \forall \{a_j, ..., a_l\} \in A_i$$

Reward function (14)

$$R_i(s, a_j) := R_0(s, a_j), \;\; \forall s \in S_0, \;\; \forall \{a_j, ..., a_l\} \in A_i$$

Parameterization of the *abstract actions* is the main motivation for adopting a bottom-up procedure during planning. A key insight is the fact that the algorithm uses the *local policy* learned for subtask P_m when modelling the corresponding abstract action $\bar{a}_m$ in the context of the higher level subtask. Going back to the example in Figure 1, the goal is to first learn a local policy for subtask $P_{Contact}$, and then use this policy to infer model parameters for action $\bar{a}_{Contact}$, such that it is then possible to proceed and apply SOLVE(P_{Remind}). Equations 15-17 describe the exact procedure for inferring those parameters.

Transition probabilities (15)

$$T_i(s, \bar{a}_m, s') := T(s, \pi_{P_m}(s), s'), \;\; \forall (s, s') \in S_0, \; \forall \bar{a}_m \in A_i$$

Observation probabilities (16)

$$O_i(s, \bar{a}_m, o) := O(s, \pi_{P_m}(s), o), \;\; \forall s \in S_0, \;\; \forall o \in \Omega_0, \; \forall \bar{a}_m \in A_i$$

Reward function (17)

$$R(s, \bar{a}_m) := R(s, \pi_{P_m}(s)), \;\; \forall s \in S_0, \; \forall \bar{a}_m \in A_i$$

An important assumption of our approach, implicit to the above discussion of local policy optimization, is that each subtask must contain some discriminative reward information. This means that we cannot have a uniform reward function over all actions and/or states, otherwise local policies could not be meaningfully optimized. This is a common assumption in hierarchical reinforcement learning (Dietterich 2000), and the assumption is easily met in the robot control domain, where the cost of accomplishing various tasks naturally provides the necessary local reward structure. In domains where this assumption is not met, we can (as suggested in (Dietterich 2000)) introduce reward shaping to provide subtasks with the required local reward information.

State and Observation Abstraction

The hierarchical algorithm, as described so far, proposes to reduce the computational overhead of POMDP planning by partitioning the action set. In terms of computational complexity, the number of linear pieces representing an exact POMDP value function is recursively given by: $|\Gamma_t| = O(|A||\Gamma_{t-1}|^{|\Omega|})$, and can be enumerated in time: $O(|S|^2|A||\Gamma_{t-1}|^{|\Omega|})$. Therefore our hierarchical POMDP algorithm, by solving subtasks with reduced action sets, can significantly reduce the computational complexity of computing POMDP policies. However these savings are partially offset by the fact that we now have to compute many policies, instead of just one. We now discuss how in many domains it is possible to further reduce computational costs by also applying state and observation abstraction. The key idea is to define reduced state and observation sets for each subtask and apply planning using this smaller representation.

The formal conditions under which to apply exact state and observation abstraction are directly related to the model parameters. We consider a POMDP subtask P_i, with action set A_i, state set S_i and observation set Ω_i, where the state set is spanned by state features $X_i = \{X_1, X_2, \ldots, X_p\}$ and the observation set is spanned by features $Z_i = \{Z_1, Z_2, \ldots, Z_q\}$. We now consider a case where the state features can be divided into two disjoint sets, X_+ and X_-, and similarly observation features can be divided into two disjoint sets, Z_+ and Z_-. We say that state features X_- and observation features Z_- are irrelevant to subtask P_i if $\forall a_j \in A_i$:

$$\begin{aligned} R(X_+, X_-, a_j) &= R(X_+, a_j) \qquad (18) \\ P(X'_+, X_- \mid X_+, X_-, a_j) &= P(X'_+ \mid X_+, a_j) P(X'_- \mid X_+, X_-, a_j) \\ P(Z_+, Z_- \mid X_+, X_-, a_j) &= P(Z_+ \mid X_+, a_j) P(Z_- \mid X_-, a_j) \end{aligned}$$

Figure 2 illustrates these constraints in the form of a dynamic belief network. In essence, we see that state features in set X_- have no effect on the reward function, and furthermore provide no transition or observation information regarding those features (namely X_+) that do have an effect on the reward function. Consequently, state features X_- and observation features Z_- can have no effect on the value function, and therefore can be safely ignored. It is important to note that the feature irrelevance condition must hold for all actions (primitive and abstract) in a given subtask.

For general POMDP planning, applying abstraction is equivalent to finding a minimum-size representation for the problem, but once the problem is specified there is little opportunity for further abstraction. In the context of hierarchical planning however, abstraction can be applied independently to

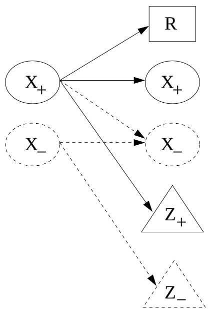

Figure 2: Dynamic belief network illustrating state/observation abstraction conditions (dashed line indicates state/observation features which can safely be ignored

each subtask—thus reducing $|S|$ and $|\Omega|$—without influencing the policy optimization any further than what is attributed to the action decomposition. The overall resulting computational savings can be tremendous (several orders of magnitude).

It is important to note that state and observation abstraction is a natural consequence of the action decomposition. In fact we observe that action decomposition and state/observation abstraction often work hand in hand. In general, a domain with a structured state set will lend itself to hierarchical planning without significant performance loss. Furthermore the same structure that gives rise to a good action decomposition often allows substantial state and observation abstraction.

A Real-World Application Domain

In this section, we present results from the real-world implementation and testing of our hierarchical POMDP robot controller in the context of an interactive robot-human task. This implementation was conducted as part of a larger project dedicated to the development of a prototype nursing assistant robot. The overall goal of the project is to develop personalized robotic technology that can play an active role in providing improved care and services to non-institutionalized elderly people. The target user is an elderly individual with mild cognitive and/or physical impairment.

Robot Platform

The robot Pearl (shown in figure 3 interacting with some users) is the primary test-bed for the behavior management system. The robot uses a differential drive system and is equipped with two on-board Pentium PCs, wireless Ethernet, a SICK laser range finder, sonar sensors, a microphone for speech recognition, speakers for speech synthesis, a touch-sensitive graphical display, actuated head unit, and stereo camera system.

On the software side, the robot features off-the-shelf autonomous mobile robot navigation system (Burgard *et al.* 1999; Thrun *et al.* 2000a), speech recognition software (Ravishankar 1996), speech synthesis software (Black, Talor, & Caley 1999), fast image capture and compression software for online video streaming, and face detection tracking software (Rowley, Baluja, & Kanade 1998). A final software component is a prototype of a flexible reminder system using advanced planning and scheduling techniques (McCarthy & Pollack 2002). For information on additional research involving this robot, the reader is referred to (Montemerlo *et al.* 2002).

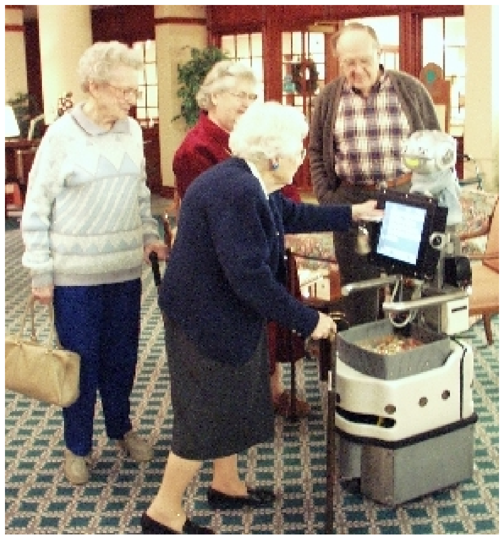

Figure 3: Pearl, the robotic nursing assistant, interacting with elderly users at a nursing facility

The robot's environment is a retirement resort located in Oakmont, PA. All experiments so far primarily involved people with relatively mild cognitive, perceptual, or physical inabilities, though in need of professional assistance.

Task description

From the many services such a robot could provide (see (Engelberger 1999; Lacey & Dawson-Howe 1998)), the work reported here has focused on the task of reminding people of events (e.g., appointments) and guiding them through their environment. At present, nursing staff in assisted living facilities spends significant amounts of time escorting elderly people walking from one location to another. The number of activities requiring navigation is large, ranging from regular daily events (e.g., meals), appointments (e.g., doctor appointments, physiotherapy, hair cuts), social events (e.g., visiting friends, cinema), to simply walking for the purpose of exercising. Many elderly people move at extremely slow speeds (e.g., 5 cm/sec), making the task of helping people around one of the most labor-intensive in assisted living facilities. Furthermore, the help provided is often not of a physical nature, as elderly people usually select walking aids over physical assistance by nurses, thus preserving some independence. Instead, nurses often provide important cognitive help, in the form of reminders, guidance and motivation, in addition to valuable social interaction.

The particular task we selected requires the robot to navigate to a person's room, alert them, inform them of an upcoming event or appointment, and inquire about their willingness to be assisted. It then involves a lengthy phase where the robot guides a person, carefully monitoring the person's progress and adjusting the robot's velocity and path accordingly. Finally, the robot also serves the secondary purpose of providing information to the person upon request, such as information about upcoming community events, weather reports, TV schedules, etc.

From an AI point of view, several factors make this task a challenging one. In addition to the well-developed topic of robot navigation (Kortenkamp, Bonasso, & Murphy 1998), the task involves significant interaction with people. The robot Pearl interacts mainly through speech and visual displays. When it comes to speech, many elderly have difficulty understanding even simple sentences, and more importantly, articulating an appropriate response in a computer-understandable way. Those difficulties arise from perceptual and cognitive deficiencies, often involving a multitude of factors such as articulation, comprehension, and mental agility. In addition, people's walking abilities vary drastically from person to person. People with walking aids are usually an order of magnitude slower than people without, and people often stop to chat or catch breath along the way. It is therefore imperative that the robot adapts to individual people—an aspect of people interaction that has been poorly explored in AI and robotics. Finally, safety concerns are much higher when dealing with the elderly population, especially in crowded situations (e.g., dining areas.)

POMDP Modelling

The robot interface domain for the selected task was modelled using 576 states, which are described using a collection of multi-valued state features. Those states were not directly observable by the robot interface manager; however the robot was able to perceive 18 distinct observations. The state and observation features are listed in table 3.

Observations were perceived through 5 different modalities; in many cases the listed observations constitute a summary of more complex sensor information. For example, in the case of a user-emitted speech signal, a keyword filter was applied to the output of the speech recognizer (e.g. *"Give me the weather forecast for tomorrow."* $\rightarrow$ *SpeechKeyword=weather*); for the laser sensor, the raw laser data was processed and correlated to a map to determine when the robot had reached a known landmark (e.g. $\rightarrow$ *Laser=robotAtHome*.) In general the speech recognition and touchscreen input were used as redundant sensors to each other, passing in very much the same information, but assumed to have a greater degree of reliability when coming from the touchscreen. The *Reminder* observations were received from a high-level intelligent scheduling module (see (McCarthy & Pollack 2002) for details about this component.)

In response to the observations, the robot could select from 19 distinct actions, falling into three broad categories:

- Communicate={RemindPhysioAppt, RemindVitamin, UpdateChecklist, CheckUserPresent, TerminateGuidance, TellTime, TellWeather, ConfirmGuideToPhysio, VerifyInfoRequest, ConfirmWantTime, ConfimWantWeather, ConfirmGoHome, ConfirmDone}
- Move={GotoPatientRoom, GuideToPhysio, GoHome}
- Other={DoNothing, RingDoorBell, RechargeBattery}

Each discrete action enumerated above invoked a well-defined sequence of operations on the part of the robot (E.g. *GiveWeather* $\rightarrow$ *SpeechSynthesis="Tomorrow's weather should be sunny, with a high of 80."*.) The actions in the *Communicate* category involved a combination of redundant speech synthesis and touchscreen display, where the

State features	*Feature values*
RobotLocation	home, room, physio
UserLocation	room, physio
UserPresent	yes, no
ReminderGoal	none, physio, vitamin, checklist
UserMotionGoal	none, toPhysioWithRobot
UserInfoGoal	none, wantTime, wantWeather

Observation features	*Feature values*
Speech	yes, no, time, weather, go, unknown
Touchscreen	t_yes, t_no, t_time, t_weather, t_go
Laser	atRoom, atPhysio, atHome
Reminder	g_none, g_physio, g_vitamin, g_checklist

Table 3: Component description for human-robot interaction scenario

selected information or question was presented to the user through both modalities simultaneously. Given the sensory limitations common in our target population, we found the use of redundant audio-visual important for both communication to and from the robot. The actions in the *Move* category were translated into a sequence of motor commands by a motion planner, which uses dynamic programming to plan a path from the robot's current position to its destination.

The POMDP model parameters were selected by a designer. The reward structure, also hand-crafted, reflects the relative costs of applying actions in terms of robot resources (e.g. robot motion actions are typically costlier than spoken verification questions), as well as reflecting the appropriateness of the action with respect to the state. For example, we use:

- positive rewards for correctly satisfying a goal
  E.g. R($a_i = TerminateGuidance$)=+50
  if $s_i(UserMotionGoal) = \{toPhysioWithRobot\}$
  and $s_i(RobotLocation) = \{physio\}$
  and $s_i(UserLocation) = \{physio\}$;
- a large negative rewards for applying an action unnecessarily
  E.g. R($a_i = GuidetoPhysio$)=-200
  if $s_i(UserMotionGoal) = \{none\}$;
- a small negative reward for verification questions
  E.g. R($a_i = ConfirmGuidetoPhysio$)=-2
  given any state condition.

and so on. This reward function satisfies the local reward assumption for each subtask, assuming the action hierarchy proposed in figure 1.

Finally, we also applied state/observation abstraction to this domain, following the conditions specified in figure 2. We were thus able to eliminate between one and five state features for each subtask. In general, lower-level subtasks yielded more abstraction (e.g. subtask P_{Inform} required only the *UserInfoGoal* state feature and the {*speech*, *touch*} observation features), whereas higher subtasks required access to most state features.

Experimental Results

The planning problem described here is far beyond the reach of exact POMDP solvers. Using the hierarchical POMDP, the high-level decision making problem in Pearl is tractable, and a near-optimal control policy can be computed off-line. Thus,

Observation	True State	Action	Reward
pearl hello	request_begun	say_hello	100
pearl what is like	start_meds	**ask_repeat**	-100
pearl what time is it *for will the*	want_time	say_time	100
pearl was on abc	want_tv	ask_which_station	-1
pearl was on abc	want_abc	say_abc	100
pearl what is on nbc	want_nbc	**confirm_channel_nbc**	-1
pearl yes	want_nbc	say_nbc	100
pearl go to the that pretty good what	send_robot	ask_robot_where	-1
pearl that that hello be	send_robot_bedroom	**confirm_robot_place**	-1
pearl the bedroom any i	send_robot_bedroom	go_to_bedroom	100
pearl go it eight a hello	send_robot	ask_robot_where	-1
pearl the kitchen hello	send_robot_kitchen	go_to_kitchen	100

Table 4: An example dialog with a test subject. Actions in bold font are clarification actions, generated by the POMDP because of high uncertainty in the speech signal.

during execution time the controller simply monitors the state (calculates the posterior) and looks up the appropriate control.

An important property of the computed POMDP policy is the inclusion of information-gathering actions. These actions have the specific purpose of gathering state-disambiguating data, as opposed to getting closer to the goal. In the domain described here, information-gathering actions are used to clarify or confirm the user's intent.

Table 4 shows an example dialog between the robot and a test subject (using a different task domain, developed for earlier in-lab experiments.) Because of the uncertainty management in POMDPs, the robot chooses to ask a clarification question at three occasions. The number of such questions depends on the clarity of a person's speech, as detected by the Sphinx speech recognition system.

In the nursing home environment, we tested the robot in five separate experiments, each lasting one full day. The first three days focused on open-ended interactions with a large number of elderly users, during which the robot interacted verbally and spatially with elderly people with the specific task of delivered sweets. This allowed us to gauge people's initial reactions to the robot.

Following this, we performed two days of formal experiments using the exact domain described in table 3. During these experiments, the robot autonomously led 12 full guidances, involving 6 different elderly people. Figure 4 shows an example guidance experiment, involving an elderly person who uses a walking aid. The sequence of images illustrates the major stages of a successful delivery: from contacting the person, explaining to her the reason for the visit, walking her through the facility, and providing information after the successful delivery—in this case on the weather.

In all guidance experiments, the task was performed to completion. Post-experimental debriefings illustrated a uniform high level of excitement on the side of the elderly. Overall, only a few problems were detected during the operation. None of the test subjects showed difficulties understanding the major functions of the robot. They all were able to operate the robot after less than five minutes of introduction. However, initial flaws with a poorly adjusted speech recognition system led to occasional confusion, which was fixed during the course of this project. An additional problem arose from the robot's initial inability to adapt its velocity to people's walking pace,

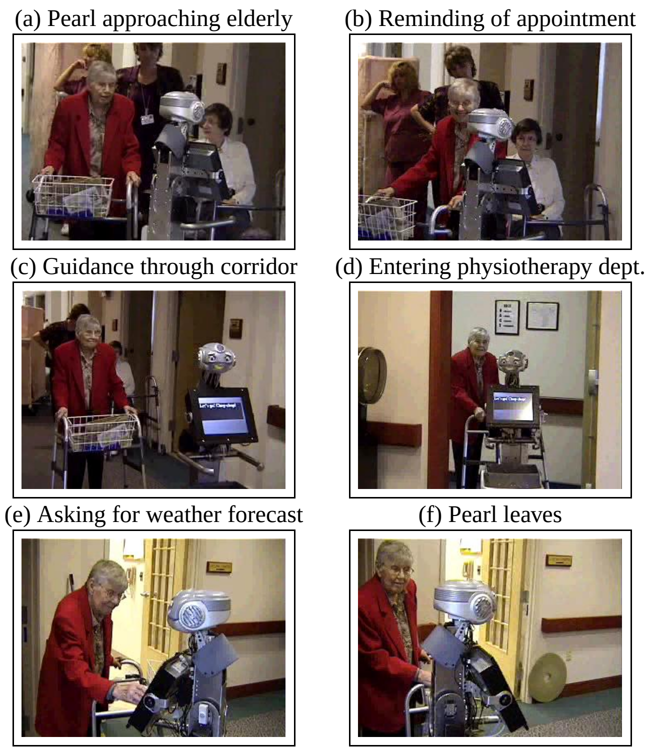
(a) Pearl approaching elderly
(b) Reminding of appointment
(c) Guidance through corridor
(d) Entering physiotherapy dept.
(e) Asking for weather forecast
(f) Pearl leaves

Figure 4: Example of a successful guidance experiment. Pearl picks up the patient outside her room, reminds her of a physiotherapy appointment, walks the person to the department, and responds to a request of the weather report. In this interaction, the interaction took place through speech and the touch-sensitive display.

which was found to be crucial for the robot's effectiveness.

We are currently engaged in experimental work that builds on the results presented here, including comparing our hierarchical POMDP approach to alternative solutions. Future experiments will include carrying out longer and more complex scenarios in the nursing home, where the robot will carry on tasks such as taking residents for walks, in support of varied physical and social activities.

Discussion

This paper describes a POMDP approach to high-level robot behavior control and presents an algorithmic approach to POMDPs which uses the structured nature of the robot planning domain to facilitate planning. The POMDP-based controller has been tested successfully in experiments in an assisted living facility.

This work demonstrates that POMDPs have matured to a level that makes them applicable to real-world robot control tasks. Furthermore, our experimental results suggest that uncertainty matters in high-level decision making. These findings challenge a long term view in mainstream AI that uncertainty is irrelevant, or at best can be handled uniformly at the higher levels of robot control (Giacomo 1998; Lakemeyer 2000). We conjecture instead that when robots interact with people, uncertainty is pervasive and has to be considered at all levels of decision making, not solely in low-level perceptual routines.

References

AAAI. 1998. Fall symposium on planning with partially observable Markov decision processes. See http://www.cs.duke.edu/˜mlittman/talks/pomdp-symposium.html.

Arkin, R. 1998. *Behavior-Based Robotics*. MIT Press.

Black, A.; Talor, P.; and Caley, R. 1999. The festival speech synthesis system. 1.4 edition.

Boyen, X., and Koller, D. 1998. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 33–42.

Brooks, R. A. 1985. A robust layered control system for a mobile robot. Technical Report TR AI memo 864, MIT.

Burgard, W.; Cremers, A. B.; Fox, D.; Hahnel, D.; Lakemeyer, G.; Schulz, D.; Steiner, W.; and Thrun, S. 1999. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*.

Cassandra, A.; Littman, M. L.; and Zhang, N. L. 1997. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, 54–61.

Cassandra, A. 1999. http://www.cs.brown.edu/research/ai/pomdp/code/.

Darrell, T., and Pentland, A. 1996. Active gesture recognition using partially observable Markov decision processes. In *Proceedings of 13th IEEE Intl. Conference on Pattern Recognition (ICPR)*.

Dayan, P., and Hinton, G. 1993. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 5, 271–278. San Francisco, CA: Morgan Kaufmann.

Dean, T., and Lin, S. H. 1995. Decomposition techniques for planning in stochastic domains. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Engelberger, G. 1999. *Handbook of Industrial Robotics*. John Wiley and Sons.

Gat, E. 1996. Esl: A language for supporting robust plan execution in embedded autonomous agents. AAAI Fall Symposium on Plan Execution.

Giacomo, G. D., ed. 1998. *AAAI Fall Symposium on Cognitive Robotics*.

Hauskrecht, M. 2000. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research* 13:33–94.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Kaelbling, L. P. 1993. Hierarchical reinforcement learning: Preliminary results. In *Machine Learning: Proceedings of the 1993 International Conference (ICML)*.

Kortenkamp, D.; Bonasso, R. P.; and Murphy, R., eds. 1998. *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press.

Lacey, G., and Dawson-Howe, K. M. 1998. The application of robotics to a mobility aid for the elderly blind. *Robotics and Autonomous Systems* 23.

Lakemeyer, G., ed. 2000. *Second International Workshop on Cognitive Robotics*.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Learning policies for partially obsevable environments: Scaling up. In *Proceedings of Twelfth International Conference on Machine Learning*.

McCarthy, C. E., and Pollack, M. 2002. A plan-based personalized cognitive orthotic. In *Proceedings of the 6th International Conference on AI Planning & Scheduling*.

McGovern, A.; Precup, D.; Ravindran, B.; Singh, S.; and Sutton, R. S. 1998. Hierarchical optimal control of MDPs. In *Proceedings of the 10th Yale Workshop on Adaptive and Learning Systems*, 186–191.

Montemerlo, M.; Pineau, J.; Roy, N.; Thrun, S.; and Verma, V. 2002. Experients with a mobile robotic guide for the elderly. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'02)*.

Nourbakhsh, I.; Powers, R.; and Birchfield, S. 1995. Dervish: An office-navigation robot. *AI Magazine* Summer:53–60.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, volume 10.

Poupart, P., and Boutilier, C. 2000. Value-directed belief state approximation for POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*.

Ravishankar, M. 1996. *Efficient Algorithms for Speech Recognition*. Ph.D. Dissertation, Carnegie Mellon.

Rowley, H. A.; Baluja, S.; and Kanade, T. 1998. Neural netowrk-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(1).

Roy, N., and Thrun, S. 2000. Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems*, volume 12.

Roy, N.; Pineau, J.; and Thrun, S. 2000. Spoken dialog management for robots. In *Proceedings of the 38th Annual Meeting of the Association for Computation Linguistics*.

Simmons, R., and Koenig, S. 1995. Probabilistic navigation in partially observable environments. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, 1080–1087.

Singh, S. 1992. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning* 8:323–339.

Sondik, E. J. 1971. *The Optimal Control of Partially Observable Markov Processes*. Ph.D. Dissertation, Stanford University.

Thrun, S.; Beetz, M.; Bennwitz, M.; Burgard, W.; Cremers, A. B.; Dellaert, F.; Fox, D.; Hähnel, D.; Rosenberg, C.; Roy, N.; Schulte, J.; and Schulz, D. 2000a. Probabilistic algorithms and the interactive museum tour-guide robot Minerva. *International Journal of Robotics Research* 19(11).

Thrun, S.; Fox, D.; Burgard, W.; and Dellaert, F. 2000b. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence* 101:99–141.