2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

Smoothed Sarsa: Reinforcement Learning for Robot Delivery Tasks

Deepak Ramachandran
Computer Science Dept.
University of Illinois at Urbana-Champaign
Urbana, IL-61801
dramacha@uiuc.edu

Rakesh Gupta
Honda Research Institute USA, Inc.
800 California Street, Suite 300,
Mountain View, CA 94041
rgupta@hra.com

Abstract— **Our goal in this work is to make high level decisions for mobile robots. In particular, given a queue of prioritized object delivery tasks, we wish to find a sequence of actions in real time to accomplish these tasks efficiently. We introduce a novel reinforcement learning algorithm called *Smoothed Sarsa* that learns a good policy for these delivery tasks by delaying the backup reinforcement step until the uncertainty in the state estimate improves. The state space is modeled by a Dynamic Bayesian Network and updated using a Region-based Particle Filter. We take advantage of the fact that only discrete (topological) representations of entity locations are needed for decision-making, to make the tracking and decision making more efficient.**

Our experiments show that policy search leads to faster task completion times as well as higher total reward compared to a manually crafted policy. Smoothed Sarsa learns a policy orders of magnitude faster than previous policy search algorithms. We demonstrate our results on the Player/Stage simulator and on the Pioneer robot.

I. Introduction

Mobile robots have now been deployed in environments with people including offices [1] and hospitals [2]. Such robots need to keep track of relevant entities (people and objects) in the environment, and perform multiple tasks all at the same time. The robot must plan efficient sequences of actions to accomplish these tasks. This planning needs to be responsive to new tasks being added dynamically and changes in the environment.

In this paper, we are concerned with dynamically planning efficient action sequences for multiple delivery tasks of the form "Deliver object X to person Y". Our primitive actions are of the kind "Move to location X", "Pick Up" and "Deliver". We assume that we have low-level controllers and navigational algorithms available to perform these actions. (though not always successfully).

The standard model for such stochastic decision-making problems is the Partially Observable Markov Decision Process (POMDP). The goal of a POMDP is to construct a *policy* for the robot that determines which action to take at any state that could arise during operation. A good policy results in a sequence of actions that maximizes a global utility criterion such as expected total reward or minimum expected time for task completion. Policies can be manually encoded, but these work well only for small instances and get progressively harder to update when new tasks are added. Policies for complex decision making tasks are usually learnt by *Reinforcement learning*(RL) [3], a framework for learning optimal behavior by trial and error.

Our key contribution is a novel reinforcement learning algorithm specialized for mobile robots called *Smoothed Sarsa*. Based on Sarsa [4], which proceeds by learning a *Q function* that approximates the value of an action taken at a state. When there is high uncertainty in the state estimator (such as when the relevant entities have not been observed), the learning step in Smoothed Sarsa is delayed until a better state estimate is obtained (e.g. after an entity is observed). Then the state estimate is *smoothed* i.e. taken back in time, to learn the value that should have been assigned to the original state. We show how to do the smoothing using particle filters and do the Q-learning such that convergence properties are preserved.

Our second contribution is the use of a quasi-topological representation of the state-space for decision making. When making decisions about what action to take next, the exact position of an object or person is of little relevance. What matters is the probability that the entity is at some approximate region, such as in front of his desk or on the kitchen table. Our entity-tracking algorithm, the *region-based particle-filter*, is specially designed to exploit this property.

POMDPs have been applied extensively in the past [5], [6], [7], [8] to robot navigation and some crude forms of decision-making. The applicability of these approaches are limited by the high dimensional discretization of space they use, hand tuning of policy, and the excessive time taken to learn a good policy. To our knowledge, ours is the first work that solves task-level decision making problems for robots at this level of generality.

We implemented our approach in simulation and on a Pioneer mobile robot in a 5 room office environment. Smoothed Sarsa managed to learn a good policy in 3 hours, whereas PEGASUS [9] took days to return a policy with worse performance. In simulated trials, the learned policy completed a set of delivery tasks 30% faster on average than a carefully crafted manual policy.

The rest of this paper is organized as follows. In section II we describe the state-space used by the decision making algorithm and the region-based particle filter. In section III, we review the theory of POMDPs and reinforcement learning. Section IV describes our Smoothed-Sarsa algorithm. Section V presents our experiments. Sections VI and VII discuss related work and conclusions respectively.

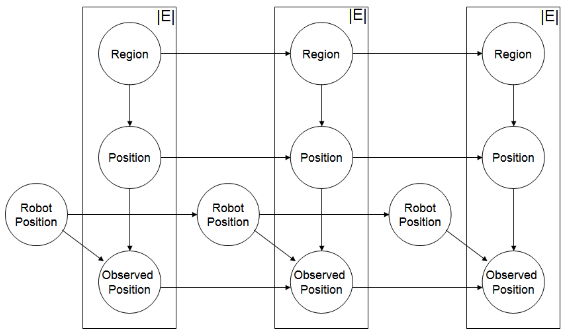

Fig. 1. The complete DBN. $|E|$ = number of entities.

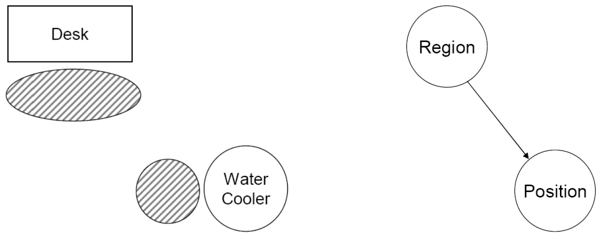

Fig. 2. Location of entities are modeled by a discrete *region* variable R, and a position, $\langle x, y\rangle$ whose distribution's parameters are conditioned on R.

II. Belief State Representation and Filter

We begin our exposition by describing the belief state maintained by the robot of its environment, which will be the state space used for planning. We specify the representation of robot pose, the region-based representation of locations of people and objects (entities), and the transition model. The temporal model incorporating all these components is a *Dynamic Bayesian Network (DBN)* (Figure 1), updated using our region-based particle filter.

A. Robot pose

Robot pose is defined by an (x, y, θ) tuple. A robot localization algorithm, such as FastSLAM [10], is used to update the robot pose using the controls and sensor readings as input. Note that entity observations depend on both true entity positions and the robot pose, since the robot pose is needed to translate the relative coordinates returned by the sensors into absolute (world) coordinates.

B. Entity Location

Entity locations are modeled by a two-layer representation as shown in Figure 2. For each entity, we have a discrete *region* variable that indicates its approximate location such as in front of the desk, or beside the water cooler. There are a small number of these regions, and each region defines a prior distribution over the exact position of the entity. For example the *position* variable, $X_t = (x_t, y_t)$, could be drawn from a Gaussian distribution

$$X_t \sim N(\mu_{R_t}, \Sigma_{R_t})$$

with mean μ_{R_t} and covariance Σ_{R_t} for a region R_t.

The advantage of this representation is that task-level decision-making only needs to consider the upper (discrete) layer of the state space. Also, domain knowledge about the environment can be incorporated into the discrete prior on regions.

C. State transitions

State transitions for each entity can occur at either the region or position level. Transitions at the region level represent deliberate movement of the entity from one region to another (e.g. a person moving from the office to the coffee machine) and happen with small probability at each time step. When a transition is made from a region $R_t = r$ to another $R_{t+1} = r'$, the new position X_{t+1} is drawn from the prior for r'. These probabilities can be learned from observations or hard-coded from domain knowledge (our current approach).

Transitions at the position level represent movement by the entity within a region (e.g. fidgeting in an office chair). We model this behavior by a *Brownian motion* process. The dynamics are chosen such that over time and in the absence of observations or region-level transitions, the particle distribution reverts to the prior for the region, corresponding to the intuition that the entity remains in the region but could be anywhere inside it. This is a simpler model than many commonly used in robotics, like those based on Extended Kalman Filters [11]. It is sufficient because we are not interested in accurate tracking of entity motions, but only their locations.

D. Region-based Particle Filters

Particle filters are used for tracking the posterior distribution of entity locations since updating cannot be done in closed form for our model. We maintain a set of samples, called *particles* (see Figure 3), from the distribution of region and position variables for each entity. These particles are updated at each time step t by the *region-based particle filter* (shown in Table I), which is based on the Bootstrap filter [12].

At each time step, the following update occurs: First, we apply the transition model for regions to each particle individually. If this causes a particle to move to a new region r', its new position is sampled from the prior for r'. Otherwise we apply the motion model for position variables (Sec. II-C). Next we re-weight each particle by the conditional likelihood of the current observation given the particle's position (These likelihoods are derived from the *observation model* of the robot).

Finally we perform a re-sampling step. A new particle set is created by sampling the previous set of particles in proportion to their weights. The crucial point here is that the re-sampling is done on a per-region basis, to keep the number of particles in a region equal to the total weight of that region. This keeps the probability mass of each region tightly controlled. Essentially, the only way for mass to shift from one region to another is through the transition model. Thus,

Algorithm 1 Region-based Particle Filter update at time step t. P_t is the particle set at time t. $p.r, p.x, p.y$ are the region, x and y positions respectively of particle p. O_t is the observation at time t.

```
for each $p^i_{t-1} \in P_{t-1}$ do
    Generate region $\overline{p^i_t}.r$ using transition model for $p^i_{t-1}.r$.
    if $\overline{p^i_t}.r \neq p^i_t.r$ then
        Generate position $(\overline{p^i_t}.x, \overline{p^i_t}.y)$ from prior for $\overline{p^i_t}.r$
    else
        Generate $(\overline{p^i_t}.x, \overline{p^i_t}.y)$ from motion model for $\overline{p^i_t}.r$
    end if
end for
Apply the Observation model. Set the weight of $\overline{p^i_t}$,
    $w_i = Pr(O_t | (\overline{p^i_t}.x, \overline{p^i_t}.y))$
for each region $r$ do
    Set $\overline{P_{t,r}} = \{\overline{p^i_t} | \overline{p^i_t}.r = r\}, W_r = \sum_{\overline{P_{t,r}}} w_i$
    Take $\lfloor W_r \rfloor$ samples from $\overline{P_{t,r}}$ such that each $\overline{p^i_t} \in \overline{P_{t,r}}$
    is chosen with prob. $\propto w_i$. Call them $P_{t,r}$.
end for
Set $P_t = \cup_r P_{t,r}$
```

the estimates of the discrete probabilities at the region layers will be highly accurate, even when the position variables are not. This improves the performance of planning algorithms that use only the discrete region values for decision-making.

In the next section we describe how decision making for delivery tasks is done in a POMDP with the entity beliefs as the state space.

III. Markov Decision Processes and Reinforcement Learning

Markov Decision Processes (MDPs) provide a mathematical framework for decision making in a stochastic environment. A Markov Decision Process (MDP) is a tuple (S, A, T, γ, R) where:

S the set of states

A is the set of actions

$T : S \times A \times S \mapsto [0, 1]$ determine the state transition probabilities

$\gamma \in [0, 1)$ is called the discount factor

$R : S \times A \mapsto \mathbb{R}$ is the reward function.

A (stationary) *policy* is a map $\pi : S \mapsto A$ and the (discounted, infinite-horizon) *value* of a policy π at state $s \in S$, denoted $V^\pi(s)$, is :

$$V^\pi(_1) = E_{s_1,s_2,\ldots}[R(s_1, \pi(s_1)) + \gamma R(s_2, \pi(s_2)) + \ldots |\pi]$$

where $Pr(s_{i+1}|s_i, \pi) = T(s_i, \pi(s_i), s_{i+1})$ and E is the expectation operator. We wish to find an *optimal policy* π^* such that $V^\pi(s)$ is maximized for all $s \in S$ by $\pi = \pi^*$. We also define the following auxiliary *Q-function*:

$$Q^\pi(s_t, a_t) = R(s_t, a_t) + \gamma E_{s_{t+1} \sim T(s_t,a_t,\cdot)}[V^\pi(s_{t+1})] \quad (1)$$

We denote the Q-function of the optimal policy π^* as Q^*.

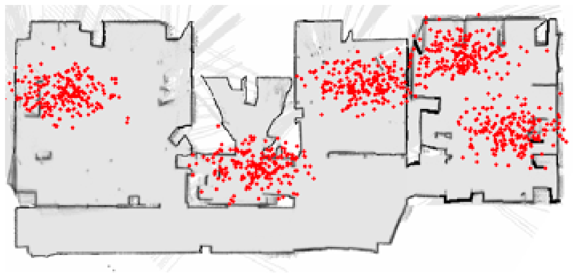

Fig. 3. A snapshot from the particle filter at time step t. Each cluster of particles indicates a region.

Note that from Q^*, the policy π^* can be recovered as :

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} Q^*(s, a) \quad (2)$$

In *reinforcement learning*, an agent is placed in an MDP and allowed to take actions that move it from state to state. It observes the sequences of states visited and the reward returned at each state, but otherwise has no knowledge of the MDP parameters. Its goal is to determine a policy with value as close as possible to the optimal policy.

A. POMDPs

Often the agent executing the MDP does not have direct knowledge of the current state. For example, in our delivery task application, the robot may not know at the current time step where the relevant entities are, but it might have a belief distribution over their locations. In such a case an extension to MDPs called the *Partially Observable Markov Decision Process* (POMDP) can be used. A POMDP is a 6-tuple (S, B, A, T, γ, R), where in addition to the MDP elements we have a set of *belief states* B, each of which is a probability distribution over the set of states S. The policy is now a function from the belief state to the action space, $\pi : B \to A$.

POMDPs are notorious difficult to solve even for small problems. Many specialized solution techniques have been proposed such as PEGASUS [9]. One common approach is to regard the POMDP as an MDP with the belief space B as the underlying state space (the *Information-state MDP* [13]). Our algorithm will be in this spirit. In particular, we define Q functions over the belief space. For every $b_t \in B$,

$$\begin{aligned} Q^\pi(b_t, a) &= E_{s_t \sim b_t}[Q^\pi(s_t, a)] \\ &= R(b_t, a) + \gamma E_{b_{t+1} \sim T(b_t,a,\cdot)}[V^\pi(b_{t+1})] \end{aligned}$$

Note that we have extended the definition of T, R, Q and V to the belief state by taking expectations in the obvious way.

B. Q function approximation

Our reinforcement learning approach will be to learn the optimal Q^* and then use equation 2 to derive π^*. We shall represent the Q function in the following parametrized form:

$$Q^\pi(s, a) \approxeq \sum_{i=1}^{n} w_i^\pi \phi_i(s, a) = \boldsymbol{w}^\pi \cdot \boldsymbol{\phi}(s, a) \quad (3)$$

Algorithm 2 Smoothed-Sarsa with Function Approximation.

```
Input: initial belief state $b_0$, arbitrary action $a_0$.
$\boldsymbol{w} := \mathbf{0}$
for $t = 0, 1, 2, \ldots$ until $\boldsymbol{w}$ converges do
    Take action $a_t$, observe $r_t, b_{t+1}$.
    $a_{t+1} := \text{argmax}_a\, Q(b_{t+1}, a)$.
    Put $(t, b_t, a_t, r_t, a_{t+1})$ onto BACKUPQUEUE.
    For each $(t', b_{t'}, a_{t'}, r_{t'}, a_{t'+1})$ on BACKUPQUEUE (in order)
    if $var(b^t_{t'}) \leq \theta_{\text{thresh}}$ or $t > t' + k_{\max}$ then
        $\delta^t_{t'} := r_{t'} + \gamma Q(b^t_{t'+1}, a_{t'+1}) - Q(b_{t'}, a_{t'})$
        $\boldsymbol{w} \overset{\alpha}{\leftarrow} \delta^t_{t'} \nabla_{\boldsymbol{w}} Q(b_t, a_t)$
        Remove $(t', b'_t, a'_t, r'_t, a'_{t+1})$ from BACKUPQUEUE.
    end if
end for
Return $\boldsymbol{w}$
```

where $\boldsymbol{\phi} = \phi_1, \phi_2, \ldots, \phi_n$ is a vector of fixed feature functions and $\boldsymbol{w}^\pi = w^\pi_1, w^\pi_2, \ldots, w^\pi_n$ is a vector of weights. The feature functions are chosen to encode properties of the state and action that are relevant to determining the Q value (e.g. for the delivery task, the distance to each region or probability of an object being in a region). We assume that the true Q^* can be well-approximated by some weighted sum of these feature functions and thus the reinforcement learning problem reduces to finding the optimal weights, $\boldsymbol{w}^*$, s.t. $Q^*(s, a) \approxeq \boldsymbol{w}^* \cdot \boldsymbol{\phi}(s, a)$, for all $s \in S, a \in A$.

We can extend this approximation to a function of belief state b in the obvious way:

$$Q^*(b, a) = E_{s\sim b}[Q^*(s, a)] = \boldsymbol{w}^* \cdot E_{s\sim b}[\boldsymbol{\phi}(s, a)] = \boldsymbol{w}^* \cdot \boldsymbol{\phi}(b, a)$$

In order to recover the optimal policy π^*, we solve

$$\pi^*(b) = \underset{a}{\text{argmax}}\, Q^*(b, a) = \underset{a}{\text{argmax}}\, [\boldsymbol{w}^* \cdot \boldsymbol{\phi}(b, a)]$$

For our problem, we consider a finite set of actions with at most one or two parameters so this maximization can be easily done.

IV. The Smoothed-Sarsa Algorithm

Smoothed-Sarsa (Algorithm 2) is a reinforcement learning algorithm based on Sarsa ([4]). The main feature of Sarsa is the technique of *bootstrapping*: After an action a_t is taken and the robot moves from belief state b_t to a new state b_{t+1} while collecting reward r_t, an action a_{t+1} is chosen according to the current policy, and the predicted Q value of the new state is used to compute an improved estimate for the previous state's Q value, called the *backup target*:

$$\delta_t = r_t + \gamma Q(b_{t+1}, a_{t+1}) - Q(b_t, a_t)$$

Observe that if $Q = Q^*$, then $E[\delta_t] = 0$.

The backup is used to adjust the weights of the policy after each step as follows:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \delta_t \nabla_{\boldsymbol{w}} Q(b_t, a_t)$$

where α is the *learning rate*, set to some small value and then decreased over time. This backup step is repeated every time an action is taken, until the weights converge. In many applications of Reinforcement Learning, aggressive use of backups has been shown to make the difference between a working and non-working policy [3].

However, the problem with the Sarsa update step is that the backed-up estimate of $Q(b_{t+1}, a_{t+1})$ can be very poor if the uncertainty in the state estimate is high. In the delivery task example, consider a situation where the robot has picked up an object and is trying to determine the Q-value of a move action. This will essentially be an expectation over the person's location and if the person has not been observed before, could be very different from the "true" Q of the underlying state. Ideally, one would like to use $Q(s_{t+1}, a_{t+1})$ in the backup, but this is unknown while learning.

The key idea of Smoothed Sarsa is to delay the backup until some time step $t + k (k > 1)$ when the uncertainty in the state estimate is reduced, and then go back in time (*smoothing*) to get a better prediction of $Q(s_{t+1}, a_{t+1})$. More precisely, define

$$b^{t+k}_{t+1} = Pr(s_{t+1} | b_{t+k})$$

the belief distribution over s_{t+1} conditioned on our belief state at time step $t+k$ (which incorporates observations after time step t). The backup in Smoothed-Sarsa is then:

$$\delta^{t+k}_t = r_t + \gamma Q(b^{t+k}_{t+1}, a_{t+1}) - Q(b_t, a_t)$$

Suppose $var(b^{t+k}_{t+1}) \ll var(b_{t+1})$. This can happen, for example, after an observation is made. Then $var(\delta^{t+k}_t) < var(\delta_t)$ and hence the convergence of the iterative procedure is correspondingly faster.

In practice, we define a threshold θ_v for the variance and a limit $k_{\max}$ for the time step delay. If the variance of b^{t+k}_{t+1} falls below θ_v or $t + k_{\max}$ time steps are completed without having done a backup for time t, then the backup is completed using $b^{t+k_{\max}}_{t+1}$. If multiple time steps go by with delayed backups, then the algorithm has to be careful to perform the backups in the right order (See Table II).

A. Smoothing

It remains to show how the smoothing is done to infer b^{t+k}_{t+1} from b_{t+k}. This can be easily done by storing the history of the region variable $(r_1, r_2, \ldots, r_{t+k})$ in each particle p_i (Recall section II-D). As new observations are made, the particles are automatically re-weighted by their likelihood so that at time $t+k$, the density of particles reflect the posterior distribution conditioned on all observations. Thus we have the smoothed belief for each entity,

$$b^{t+k}_{t+1}(r) \approxeq \sum_{p_i \in P_{t+k}} I(p_i.r_{t+1} == r)$$

where I denotes the indicator function. The total belief state will then be the product of each entity's smoothed belief.

V. Experiments

We have applied our methods to robot delivery tasks in simulation and on a real robot. We considered n-object n-person delivery scenarios in a 5-room office environment, with varying n. Reinforcement learning was done in the simulation over the course of thousands of scenarios to learn a good policy for this task which was transferred to a real robot. The actions available to the robot were:

1) Move towards region r for t seconds.
2) Pickup object x from region r.
3) Deliver currently held object to person x in region r.

The robot gets a reward for completing each task proportional to its priority. The feature vector used to approximate the Q function (Equation 3) was composed of relevant state variables such as $Pr(\text{Entity } e \text{ in Region } r)$, $\text{distance}(\text{entity } e, \text{ region } r)$, $\text{IsVisible}(\text{region } r, \text{region } r')$ and $\text{Entropy}(\text{region } r)$. We also used some non-linear functions of these basic features (such as $\frac{Pr(\text{Entity } e \text{ in Region } r)}{\text{distance}(\text{entity } e, \text{ region } r)}$) to capture any non-linear dependence the Q^* function has on the state.

Even with just two delivery tasks we can expect very complex behaviour from the optimal policy. Suppose for example, the robot was in a position to immediately pick up either of the two objects. It would generally prefer the task for which its belief about the person's location was more certain in the current state. But this could change if one task had greater priority than the other or if choosing the other task meant that unexplored regions of the environment could be visited and new observations made. When choosing its next action, the robot must not only consider regions that are likely to have relevant entities, but also the distances to them and the visibility of other regions from there. We shall show that the learned policy exhibits these behaviours, and finds near-optimal trade-offs between them.

A. Simulation

We used Smoothed Sarsa to learn a policy for the multiple delivery problem in the Player/Stage simulator [14]. We generated thousands of scenarios with varying region parameters and entity locations. The simulated robot uses a laser range-finder for localization and a blob detector to detect and recognize entities.

Using the Region-based Particle Filter for entity tracking and Smoothed Sarsa for reinforcement learning, we converged on an approximately optimal policy in 3 hours with 25,000 simulations. In comparison running the state of the art POMDP solver PEGASUS [9] on the same problem took 2 days to terminate and could only find a successful policy for single delivery tasks while ordinary Sarsa never converged at all.

In Figures 5 and 6 we compare the qualitative performance of the policy learned by Smoothed Sarsa and a manually crafted baseline policy. The manual policy does each task in order of priority, and when searching for an entity, it visits the regions in order of maximum likelihood. The smoothed-sarsa policy demonstrates more intelligent behaviour than the manual one. The manual policy starts out looking for

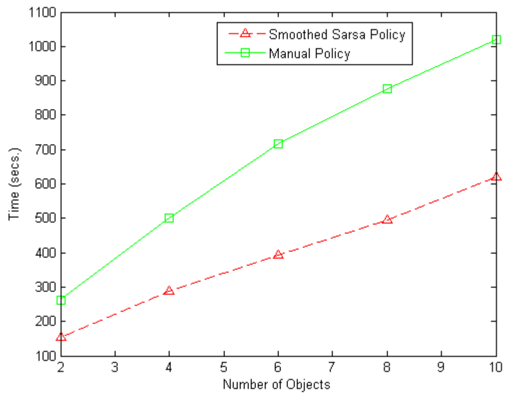

Fig. 4. Mean time to completion of tasks for Smoothed Sarsa vs. Manual Policy

the first object in the leftmost room and keeps looking for it without switching tasks after it sees the second object. Consequently it must move back and forth between the two ends of the office 3 times to complete all the tasks. The smoothed sarsa policy moves to the rightmost room first because it recognizes that the probability of completing *some* task is greatest there. It decides to do the second delivery task since both the object and the person are in the same room. It then completes the first task, moving across to the left room just once. The total run time was 50 seconds for the sarsa policy and 118 seconds for the manual policy.

In Figure 4, we show the mean performance of Smoothed Sarsa and the manual policy against the number of delivery tasks. These numbers were averaged over 100 trials. Smoothed Sarsa consistently performs better.

B. Robot

For our experiments we used a Pioneer mobile robot with 3 sensors: a SICK laser rangefinder for localization, a Swiss-ranger depth camera for person detection and a Ueye vision camera for object recognition. We used PLAYER [14] to control the robot. The person detector uses the LASIK library [15] and objects are detected by matching patterns imprinted on the object. The policy learned in simulation was transferred to the robot with no modification (the implementation of action primitives and the observation model had to be changed). We tested the policy in an office environment with the same map as in simulation. A video of the robot executing the policy for 2 tasks is part of this submission. Frames from this video are shown in Figure 7. The environment is quite cluttered with other objects and persons. People in these images are stationary for simplicity but could be moving while the robot is operating. The robot does not physically pick up and deliver objects but indicates by beep sounds when it would execute such actions if it had an actuator.

VI. Related Work

There have been other attempts at high level planning using POMDPs dealing with both noisy speech input and path planning. In particular, Pineau et al. [2] built a high level control and dialog management system using a robot and person location, person's speech commands, and goals (motion, reminders, information). They learnt a policy using a hierarchical decomposition of POMDPs that asked for confirmations to reduce uncertainty. However they made a strong assumption that the domain possesses structure that can be expressed via an action hierarchy. Schmidt-Rohr et al. [16], [17] built a POMDP with 200 states and 11 actions for a fetch and delivery task. However, their policy is manual and not learnt. Spaan et al. [8] perform both localization and path planning using POMDPs. They discretize a multiple room environment into a 500 position grid and performs an object delivery task within a group of static pickup grid locations. However their policy does not consider the interesting features of decision making that we do, and in fact can be thought of as solving multiple path planning problems.

The closest work in the reinforcement community to our Smoothed Sarsa algorithm is by Walsh et al. [18] on learning with delayed rewards, a setting where rewards are received by the agent only after some delay.

VII. Conclusions

We have proposed a framework for learning task-level decision making for robots. We make two contribution in this paper. Our key contribution is the *Smoothed Sarsa*, reinforcement learning algorithm, that delays the learning step until better state estimates are obtained. Smoothed Sarsa preserve convergence properties but with lower backup variance and faster convergence times.

Secondly, we introduce the *region-based particle filter* for tracking people and objects in the environment. The algorithm emphasizes the accurate tracking of the discrete region variables, which are more useful for decision making, at the expense of precise positional estimates.

We plan to extend this work to tasks other than object delivery, such as recycling and robot re-localization. The same methods should apply with appropriate modifications to the Q function representation. Currently the locations of the regions and the transition model for entities are hard-coded. We plan to explore how to learn them from clustering real-world observations of people's behaviour over time. We also defer to future work a rigorous theoretical analysis of Smoothed Sarsa's convergence bounds.

Acknowledgments

We wish to acknowledge Jongwoo Lim, Jiwoong Sim, and Youding Zhu at the Honda Research Institute for help with the object detector.

We thank Andrew Ng, Steve Gould, and Ian Goodfellow from Stanford University for use of the LASIK library for people detection. We also thank Eyal Amir and Shivaram Kalyanakrishnan for useful discussions.

References

[1] K. O. Arras and S. J. Vestli, "Hybrid, high-precision localization for mail distribution mobile system robot mops," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 1998.

[2] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards robotic assistants in nursing homes: Challenges and results," *Special issue on Socially Interactive Robots, Robotics and Autonomous Systems*, vol. 42, no. 3-4, pp. 271–281, 2003.

[3] R. Sutton and A. Barto, *Reinforcement Learning*. MIT Press, 1998.

[4] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," Cambridge University Engineering Dept., Tech. Rep. CUED/F-INFENG/TR 166, 1994.

[5] G. Theocharous and L. P. Kaelbling, "Approximate planning in POMDPs with macro-actions," in *Advances in Neural Information Processing Systems 16 (NIPS)*, 2004.

[6] S. Koenig and R. Simmons, "Xavier: A robot navigation architecture based on partially observable markov decision process models," in *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, R. B. D. Kortenkamp and R. Murphy, Eds. MIT Press, 1998, pp. 91 – 122.

[7] R. Simmons and S. Koenig, "Probabilistic robot navigation in partially observable environments," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, pp. 1080–1087.

[8] M. T. J. Spaan and N. Vlassis, "A point-based POMDP algorithm for robot planning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, April 2004.

[9] A. Y. Ng and M. Jordan, "PEGASUS: A policy search method for large MDPs and POMDPs," in *Proceedings of the Sixteenth Conference in Uncertainty in Artificial Intelligence (UAI)*, 2000.

[10] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002.

[11] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *The International Journal of Robotics Research*, vol. 24, no. 1, pp. 31–48, 2005.

[12] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[13] L. P. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[14] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *In Proceedings of the 11th International Conference on Advanced Robotics*, 2003, pp. 317–323.

[15] Goodfellow, Dreyfus, Gould, and Ng, "People detection in 3-dimensions on STAIR," 2008, unpublished Manuscript.

[16] S. R. Schmidt-Rohr, S. Knoop, M. Lsch, and R. Dillmann, "Reasoning for a multi-modal service robot considering uncertainty in human-robot interaction," in *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, 2008, pp. 249–254.

[17] S. R. Schmidt-Rohr, M. Losch, and R. Dillmann, "Human and robot behavior modeling for probabilistic cognition of an autonomous service robot," in *The 17th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, August 2008, pp. 635–640.

[18] T. J. Walsh, A. Nouri, and L. Li, "Planning and learning in environments with delayed feedback," in *In ECML-07*, 2007, pp. 442–453.

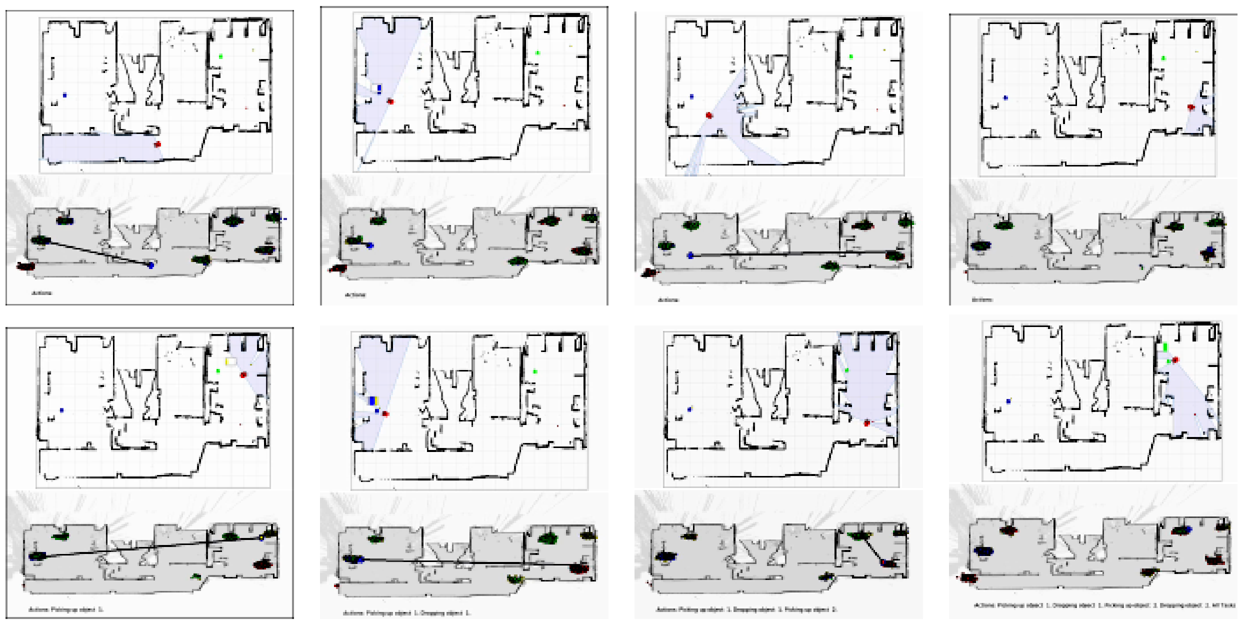

Fig. 5. Manual policy using Stage simulator. The top image in each frame is a snapshot from the simulator and the bottom is a visualization of the robot's belief state and current action. The line joining the robot to the region shows the region that the robot is moving to. Ordering from left to right (best viewed in color). TOP ROW (a) Robot looking for the first object in the left room based on prior (b) first object not found but first person found (c) Robot looking for the first object in right room (d) second object found. Robot next navigates to top part of room. BOTTOM ROW (e) First object found and picked up. Robot navigating to first person in left room (f) After delivering first object, robot navigating to second object (g) Robot picks up second object and navigates to second person in the right room (h) Second object delivered.

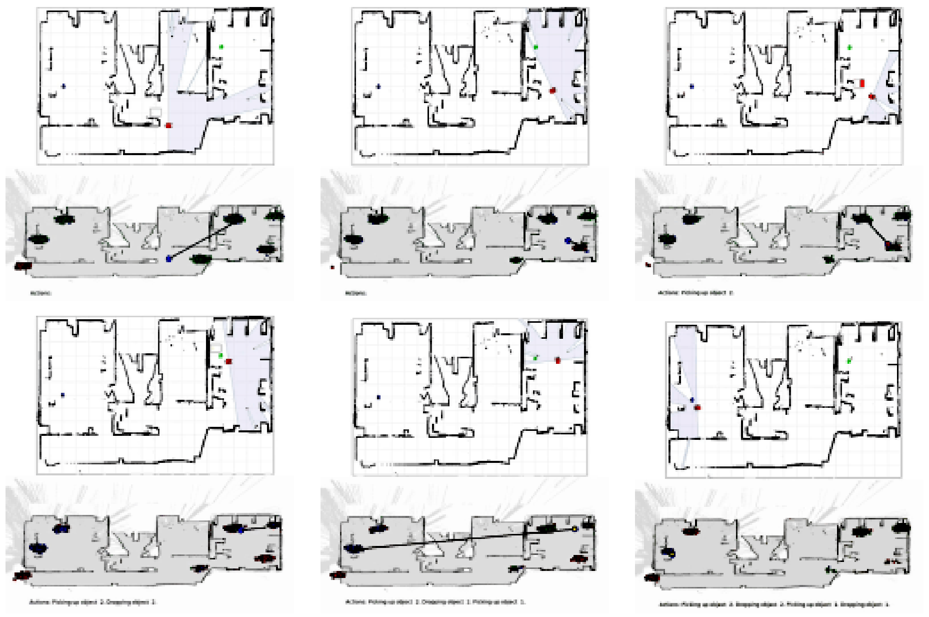

Fig. 6. Smoothed Sarsa policy using Stage simulator. The top image in each frame is a snapshot from the simulator and the bottom is a visualization of the robot's belief state and current action. The line joining the robot to the region shows the region that the robot is moving to. Ordering from left to right (best viewed in color). TOP ROW (a) Robot going to right room to look for first object because it is closer and more regions are visible (b) Robot sees second object and picks it up (c) Second object is being delivered to the second person BOTTOM ROW (d) Robot moving to the first object in the same right room (e) Robot looking for first person in the left room (f) First object delivered.

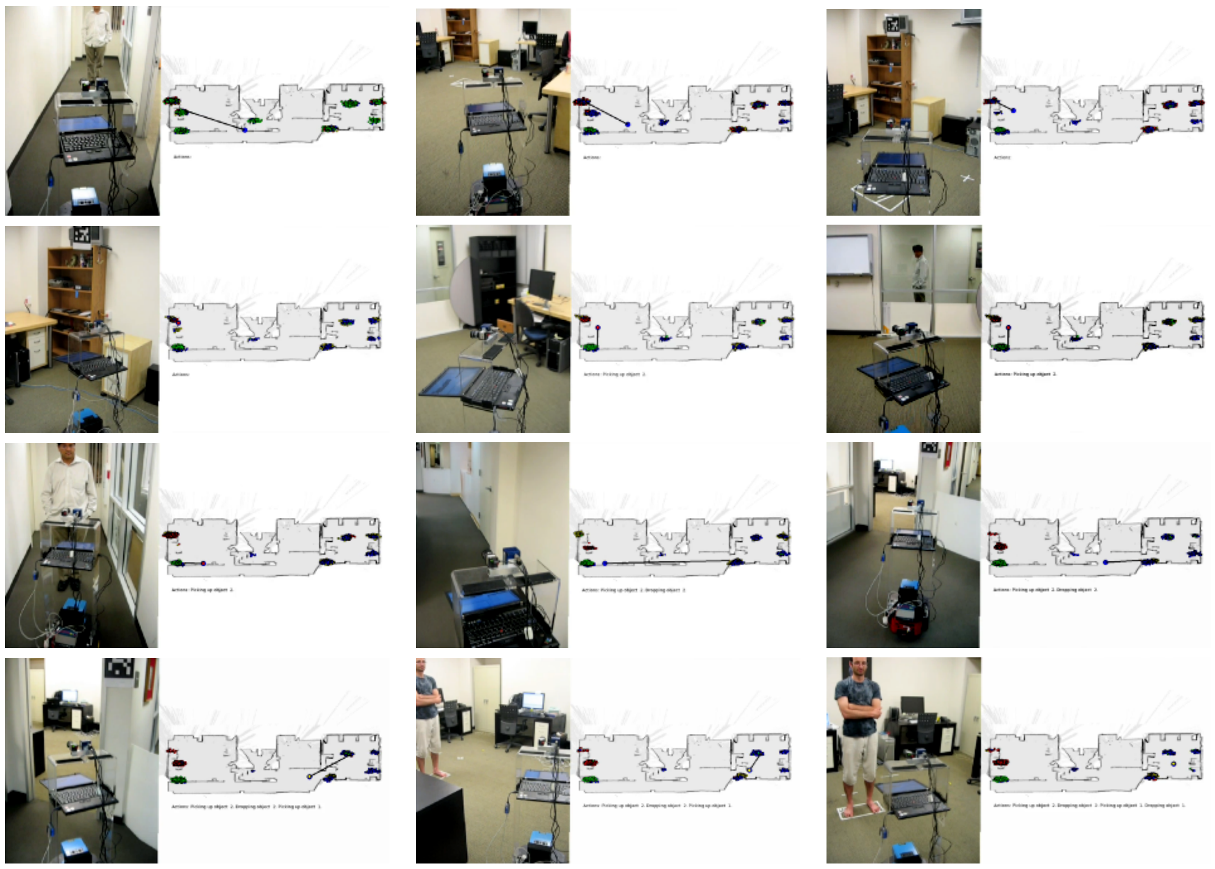

Fig. 7. Smoothed Sarsa policy on a Pioneer Robot. The line joining the robot to the region shows the region that the robot is moving to. Ordering from left to right (best viewed in color). TOP ROW (a) Robot going to left room to look for object (b) Robot looking at a different region for object (c) Robot finds the first object in the corner of the left room SECOND ROW (d) First object picked up (e) Robot navigating to the first person (f) First person as seen through the glass THIRD ROW (g) First object delivered to the first person. (h) Robot looking for the second object (i) Robot navigating to the second object. FOURTH ROW (j) Robot picks the second object and looks for second person in the right room (k) Robot navigating to person in the second room (l) Robot delivered second object to second person.