

# Hierarchical Dynamic Programming for Robot Path Planning\*

Bram Bakker and Zoran Zivkovic and Ben Kröse  
*Intelligent Autonomous Systems group, Informatics Institute*  
*University of Amsterdam*  
*Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands*  
*{bram, zivkovic, krose}@science.uva.nl*

**Abstract**— This paper addresses the question how robot planning (e.g. for navigation) can be done with hierarchical maps. We present an algorithm for hierarchical path planning for stochastic tasks, based on Markov Decision Processes (MDPs) and dynamic programming. It is more efficient than standard dynamic programming for “flat” MDPs, because it reduces the state space for all levels in its hierarchy and it allows reuse of previously computed partial policies. This computational advantage comes at the cost of some extra memory and overhead to represent and coordinate the hierarchical system, and in some cases somewhat longer paths to target locations. We demonstrate the method on artificially generated MDP data, and on real robot data from our vision-controlled robot navigating in an office environment.

**Index Terms**— Planning, Dynamic Programming, Hierarchical methods, Multiresolution methods, Mobile Robots

## I. INTRODUCTION

A number of researchers (e.g. [9], [14], [13]) have proposed to represent mobile robots’ environments using a hierarchy of maps. The central idea in all those proposals is that it makes sense to represent a large environment at multiple resolutions, or multiple levels of abstraction.

Low-level, local maps may be used to represent, for example, individual rooms in a large building in great spatial detail, without representing the spatial relationships to locations in other rooms. Such a low-level map may be used to navigate to precise target locations within an individual room, without having to worry about other rooms.

Higher-level, abstract maps may be used to represent larger areas of a building, for instance as a graph connecting rooms and corridors, without representing the exact spatial relationships of individual locations within rooms and corridors. Such a high-level map may be used to construct abstract plans to navigate from one room to another, without having to worry about exact spatial details. Also, it may be used for effective communication with humans, because the elements in the high-level map (e.g., the nodes in the graph) can be made to correspond to concepts that make sense to humans (rooms, corridors).

In sum, a hierarchy of maps may facilitate communication, map building, and planning based on the maps.

This paper addresses the question how planning (e.g. for navigation) can be done with hierarchical maps. We present

an efficient algorithm for hierarchical path planning for stochastic tasks. The maps are formalized as Markov Decision Processes (MDPs) and the planning tasks as corresponding dynamic programming problems. We describe, firstly, how a hierarchy of MDPs can be constructed, and secondly, how it can be solved using a hierarchical variation of value iteration. This approach is different from but related to both non-hierarchical [2], [3] and alternative hierarchical approaches [12], [4], [5], [1] based on MDPs, which will be discussed below.

We show that path planning done in this way can be much more efficient than when one uses one standard, monolithic, “flat” MDP, especially when the state space becomes large and when paths must be planned to many possible target locations. This computational advantage, which is particularly important in the context of real-time robot planning, comes at the cost of some extra memory and overhead to represent and coordinate the hierarchical system, and in some cases somewhat longer paths to target locations. Since our hierarchical method is based on standard MDPs, it could be extended to planning tasks other than path planning as well.

The next section briefly reviews MDPs and the corresponding standard dynamic programming solution method of value iteration. Section III describes our hierarchical MDP and dynamic programming method, and compares it to related work. Section IV describes experiments based on artificial data, illustrating the method and comparing it empirically to standard, flat dynamic programming. Section V describes similar experiments, but now on data from our real mobile robot [15]. Section VI presents conclusions and possible future work.

## II. MDPs AND DYNAMIC PROGRAMMING

For path planning in possibly stochastic domains, robot maps are commonly formalized as Markov Decision Processes (MDPs), such that the planning task becomes a dynamic programming problem [2], [3]. This formalization is appropriate for such robot planning tasks for several reasons. First of all, it takes into account the realistic assumption of uncertainty in the execution of actions (i.e. stochastic state transitions). Secondly, policies are computed for the entire state space, which is appropriate when action outcomes are uncertain. Thirdly, the resulting policies are optimal in the sense that they lead to lowest expected cost (e.g. distance traveled). Finally, it allows for straightforward inclusion of

\*This work was supported by the EU FP6-IST2020 “Cogniron” project.

cost factors other than distance traveled, such as energy consumption and obstacle avoidance (or other factors if the task goes beyond navigation).

#### A. MDPs for path planning

An MDP  $\mathcal{M}$  is a tuple  $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ .  $\mathcal{S}$  is a finite set of states  $s$ , some of which may be terminal states.  $\mathcal{A}$  is a finite set of actions  $a$ , whose availability may depend on the state.  $\mathcal{R}$  is the reward function that defines the immediate reward  $r$ . The Markov property for state and reward representations requires that the state and reward at time  $t+1$  depend only on the state and action at time  $t$ , such that the following holds:

$$\begin{aligned} p\{s_{t+1} = s, r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, \dots, s_0, a_0\} = \\ p\{s_{t+1} = s, r_{t+1} = r \mid s_t, a_t\}. \end{aligned} \quad (1)$$

Given that the Markov property holds, we can define  $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ , the state transition function that describes the probability  $p(s'|s, a)$  that the system will move from state  $s$  to  $s'$  after performing the action  $a \in \mathcal{A}$ . Successors are defined as those states  $s'$  for which  $p(s'|s, a) \neq 0$ .  $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  defines the immediate real-valued reward  $r(s, a, s')$  when action  $a$  is taken in state  $s$  and the transition to  $s'$  is made.  $r(s, a, s')$  may be a negative cost function, e.g. based on distance traveled between  $s$  and  $s'$ .

For path planning, states corresponding to target locations become terminal states. Furthermore, we assume that the set of actions in a state  $s$  simply corresponds to the set of successors  $s'$ . That is, at the current state the robot can choose from the successor states where to go next; but it only arrives at the successor state with probability  $p(s'|s, a)$ .

A policy is defined as a mapping  $\pi : \mathcal{S} \rightarrow \mathcal{A}$ . The objective is to find an optimal policy  $\pi^*$  that maximizes the expected, possibly discounted, future cumulative reward, or expected return. In the finite horizon case, i.e. when each episode ends at some time  $T$  (e.g. because a target is reached) and without discounting of future rewards, this corresponds to:

$$\begin{aligned} V^*(s) &= \max_{\pi} V^{\pi}(s) \\ &= \max_{\pi} E \left[ \sum_{t=0}^T r(s_t, \pi(s_t), s_{t+1}) \mid \pi, s_0 = s \right] \end{aligned} \quad (2)$$

for each state  $s$ . The expectation operator  $E[\cdot]$  averages over rewards and stochastic state transitions.

#### B. Value iteration

MDPs with known state transition functions and reward functions can be solved optimally using dynamic programming methods. Dynamic programming iteratively computes the value function  $V(s)$ , which represents the estimate of the expected return attainable from each state. It is guaranteed to converge to the optimal value function  $V^*(s)$ , which represents the maximum attainable expected return (eq. 2). One well-known method, value iteration, repeatedly sweeps through the state set of the MDP and, in the undiscounted, finite horizon case, updates each state's value according to

$$V(s) \leftarrow \max_a \sum_{s'} p(s'|s, a) [r(s, a, s') + V(s')] \quad (3)$$

until the largest change in value of any of the states,  $\Delta$ , is smaller than a small constant threshold. After convergence, the optimal policy is followed by simply taking the greedy action in each state:

$$\pi^*(s) = \arg \max_a \sum_{s'} p(s'|s, a) [r(s, a, s') + V^*(s')]. \quad (4)$$

### III. HIERARCHICAL MDPs AND DYNAMIC PROGRAMMING

#### A. Hierarchical MDPs

Our method extends the standard MDP framework by adding hierarchical structure. We define two types of MDPs making up a complete hierarchical system. They are both derived from a given standard, flat MDP. The first type,  $\mathcal{M}^n$ , a tuple  $\langle \mathcal{S}^n, \mathcal{A}^n, \mathcal{T}^n, \mathcal{R}^n \rangle$ , represents the given MDP at a particular level of abstraction.  $n$  indexes the level in the hierarchy,  $N$  is the number of levels in the hierarchy (decided by the designer). The given flat MDP is  $\mathcal{M}^0$ .  $\mathcal{M}^n$  for  $n \geq 1$  is constructed from  $\mathcal{M}^{n-1}$  by clustering the states in  $\mathcal{S}^{n-1}$ . A possible clustering method is briefly described below and in more detail in a companion paper [15]. Each cluster of states from  $\mathcal{S}^{n-1}$  becomes a single state in  $\mathcal{S}^n$ .

The state transition function  $\mathcal{T}^n$ , which defines  $p(s_m^n | s_k^n, a^n)$ , is constructed by determining, for all states  $s_i^{n-1} \in \mathcal{S}^{n-1}$  that belong to one cluster and correspond to state  $s_k^n$ , the cluster labels of successors  $s_j^{n-1}$  that belong to other clusters and correspond to states  $s_m^n$ . The probability  $p(s_m^n | s_k^n, a^n)$  is estimated by averaging over the corresponding probabilities  $p(s_j^{n-1} | s_i^{n-1}, a^{n-1})$ . Similarly, the reward function  $\mathcal{R}^n$ , which defines  $r(s_k^n, a^n, s_m^n)$ , is constructed by determining, for each state transition from  $s_k^n$  to  $s_m^n$ , the corresponding  $r^{n-1}(s_i^{n-1}, a^{n-1}, s_j^{n-1})$ , and averaging over them.<sup>1</sup> As before, the action set  $\mathcal{A}^n$  is defined as the set of successors  $s_m^n$  for each state  $s_k^n$ .

The second type of MDPs making up the complete hierarchical system is defined only for  $n \geq 1$  and is denoted by  $\mathcal{M}_{s_k^n, s_m^n}^{n-1}$ .  $\mathcal{M}_{s_k^n, s_m^n}^{n-1}$  is an MDP that represents the lower level ( $n-1$ ) task of navigating from higher level ( $n$ ) state  $s_k^n$  to state  $s_m^n$ . It is essentially a subset of  $\mathcal{M}^{n-1}$ , whose states are only those states  $s_i^{n-1} \in \mathcal{S}^{n-1}$  that correspond to state  $s_k^n$ , combined with those states  $s_j^{n-1} \in \mathcal{S}^{n-1}$  that are successors of states  $s_i^{n-1}$  and that correspond to state  $s_m^n$ . The states  $s_j^{n-1}$  are terminal states.  $\mathcal{A}_{s_k^n, s_m^n}^{n-1}$ ,  $\mathcal{T}_{s_k^n, s_m^n}^{n-1}$ , and  $\mathcal{R}_{s_k^n, s_m^n}^{n-1}$  follow directly from  $\mathcal{M}^{n-1}$ .  $\mathcal{M}_{s_k^n, s_m^n}^{n-1}$  is a special case intended for navigating to a specific low-level state  $s_j^{n-1}$  within a high-level state  $s_k^n$ .  $\mathcal{S}_{s_k^n, s_m^n}^{n-1}$  contains only those states  $s_i^{n-1} \in \mathcal{S}^{n-1}$  that correspond to state  $s_k^n$ , and  $s_j^{n-1}$  is the only terminal state.

Figure 1 depicts schematically how a 2-level hierarchy of MDPs is derived from a simple flat MDP using our method.

<sup>1</sup>In some cases, it may be more appropriate to be either optimistic or pessimistic about these higher level state transition probabilities and rewards, instead of simply averaging over the lower level values (e.g. see [12]).

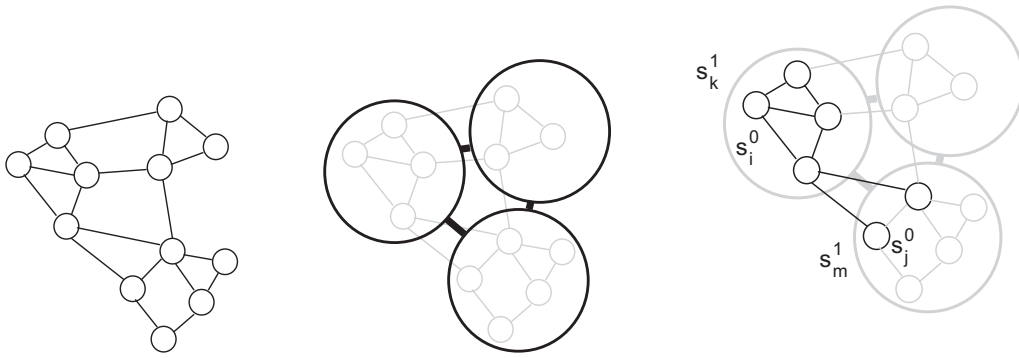


Fig. 1. Illustration of a 2-level hierarchy of MDPs. Left: The flat, low-level MDP  $\mathcal{M}^0$ . Middle:  $\mathcal{M}^1$  is constructed by clustering the states of  $\mathcal{M}^0$  and averaging over state transition probabilities and rewards. Right:  $\mathcal{M}^0_{s_k^1, s_m^1}$  is an MDP which represents the low-level task of navigating from high-level state  $s_k^1$  to state  $s_m^1$ .

### B. Hierarchical value iteration

With  $q$  states and a maximum of  $m$  admissible actions for any state, standard value iteration (for flat MDPs) requires for each sweep through the state space at most  $O(mq)$  operations in the deterministic case and  $O(mq^2)$  operations in the stochastic case. Because of this, it can become very slow with large numbers of states. Furthermore, there is no obvious way to reuse value functions obtained with previously selected target states for new target states. Our hierarchical approach has benefits on both of these issues. Thus, we use value iteration, but adapt it to our hierarchical framework.

The hierarchy of MDPs defined in the previous section allows us to efficiently compute a value function/policy for the entire state space to every possible target state of the original, flat MDP  $\mathcal{M}^0$ . For a specific low-level target state, the higher level target states in  $\mathcal{M}^n$  for all  $0 < n < N$  are determined in which this low-level target state lies. At their own levels, they become terminal states. Next, the path planning task is performed from the highest level down, using value iteration at each level. State transitions from  $s_k^n$  to  $s_m^n$  dictated by a value function at level  $n$  are modeled by the appropriate  $\mathcal{M}^{n-1}_{s_k^n, s_m^n}$  and subsequently planned, again using value iteration. In this way, the complete planning task to a low-level target state is solved recursively.

In contrast to standard, flat value iteration, when paths must be planned to multiple target states, the algorithm can in many cases reuse value functions computed for earlier target states. This is because high-level value functions remain the same as before when high-level target states do not change for this new low-level target state, and because certain high-level state transitions remain the same as before so lower-level value functions realizing those state transitions can be reused. A second advantage over standard flat value iteration is that the state spaces at all levels are reduced, leading to fewer operations per sweep through the state set and faster convergence.

Algorithm 1 provides pseudocode for the complete hierarchical planning method, assuming that a flat MDP  $\mathcal{M}^0$  is given and MDPs  $\mathcal{M}^n$  up to  $n = N - 1$ , the top level,

have already been constructed using a clustering algorithm. MDPs  $\mathcal{M}^{n-1}_{s_k^n, s_m^n}$  have not yet been constructed; they are constructed only when needed. The specific algorithm described in pseudocode assumes that the lowest level ( $n = 0$ ) target state is given by the user. The algorithm can also be used if the user provides higher-level target states, as in a command like “go to the kitchen”. In that case no target states are identified at levels below the level of the assigned high-level target state, and the algorithm ends when paths are learned that reach the high-level target state (“enter the kitchen”).

```

 $s_g^0 \leftarrow$  new target state for  $\mathcal{M}^0$ 
for all  $0 < n < N$  do
   $s_g^n \leftarrow$  determine target state for  $\mathcal{M}^n$ 
   $V^{N-1} \leftarrow \text{Solve}(\mathcal{M}^{N-1}, N - 1)$ 

Function  $\text{Solve}(\mathcal{M}, n)$ :
while  $\delta > \Delta$  (a tiny threshold) do
  for all  $s \in \mathcal{S}$  do
     $V_{new} \leftarrow \max_a \sum_{s'} p(s'|s, a)[r(s, a, s') + V(s')]$ 
    if  $|V_{new} - V(s)| > \delta$  then
       $\delta \leftarrow |V_{new} - V(s)|$ 
       $V(s) \leftarrow V_{new}$ 
if  $n > 0$  then
  for all  $s \in \mathcal{S}$  do
    if  $s = s_g^n$  then
      if  $V_{s_g^n, s_g^{n-1}}^{n-1}$  does not exist then
        Construct  $\mathcal{M}_{s_g^n, s_g^{n-1}}^{n-1}$  from  $\mathcal{M}^{n-1}$ 
         $V_{s_g^n, s_g^{n-1}}^{n-1} \leftarrow \text{Solve}(\mathcal{M}_{s_g^n, s_g^{n-1}}^{n-1}, n - 1)$ 
      else
         $s^* \leftarrow \arg \max_a \sum_{s'} p(s'|s, a)[r(s, a, s') + V(s')]$ 
        if  $V_{s, s^*}^{n-1}$  does not exist then
          Construct  $\mathcal{M}_{s, s^*}^{n-1}$  from  $\mathcal{M}^{n-1}$ 
           $V_{s, s^*}^{n-1} \leftarrow \text{Solve}(\mathcal{M}_{s, s^*}^{n-1}, n - 1)$ 
  Return  $V$ 

```

**Algorithm 1:** Pseudocode of our hierarchical value iteration algorithm.

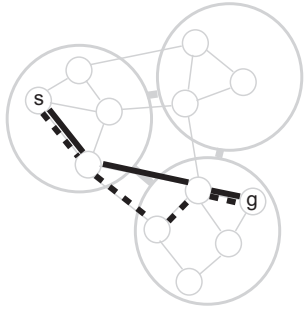


Fig. 2. Illustration of a case when our hierarchical planning method leads to longer paths from a start state  $s$  to a target state  $g$ . The solid line is the optimal path. The dashed line is the path computed by the hierarchical method, which optimally goes to the bottom high-level state, and within the bottom high-level state optimally goes to the target state.

The algorithm’s increased efficiency comes at the cost of some extra memory and overhead to represent and coordinate the hierarchical system. Furthermore, in some (but not all) cases the system can converge to somewhat longer paths to target locations than standard flat value iteration. This situation can arise because low-level value functions which are optimal with respect to reaching the next high-level state from the current high-level state are not always optimal with respect to reaching the final target state (see figure 2 for an illustration). It is also possible that high-level reward functions are not completely accurate because they average over low-level rewards. The latter problem could be remedied by updating expected rewards for  $(s_k^n, s_m^n)$  state transitions in  $\mathcal{M}^n$  with the values computed by their corresponding  $\mathcal{M}_{s_k^n, s_m^n}^{n-1}$ .

Importantly, however, given that the Markov property holds for the  $\mathcal{M}^n$  at all levels in the hierarchy ( $0 \leq n < N$ ), i.e.

$$\begin{aligned} p\{s_{t+1}^n = s^n, r_{t+1}^n = r^n \mid s_t^n, a_t^n, r_t^n, s_{t-1}^n, \dots, s_0^n, a_0^n\} = \\ p\{s_{t+1}^n = s^n, r_{t+1}^n = r^n \mid s_t^n, a_t^n\}, \end{aligned} \quad (5)$$

this hierarchical dynamic programming method is guaranteed to converge to value functions and policies which correspond to valid paths to the target location from any state from which the target location can be reached. Moreover, it converges to policies which are *optimal given the hierarchy* (cf. [5]), i.e. it will do the best it can possibly do given the pre-defined clustering of states of  $\mathcal{M}^0$ . Enforcing the Markov property at all levels means, essentially, that the clustering algorithm may not give the same cluster label to different, unconnected sets of states at any level  $0 < n < N$ . We discuss below (and more extensively in [15]) a clustering algorithm that enforces this criterion.

### C. Related work

Other studies that use some form of hierarchical planning similarly emphasize the benefit of reducing state spaces through hierarchy. These include logic-based planners [6] which use very different representations and solution methods but similarly compute abstract plans before computing

detailed plans. Furthermore, within robotics there is work on multiresolution motion planning [8] which is similar in spirit to our hierarchical planning approach but uses A\* planning techniques, does not take into account uncertainty in action outcomes, plans open loop policies from specific starting states rather than closed loop policies for all states, and does not exploit our particularly effective reuse of previously computed policies.

There is some work on other variations of hierarchical dynamic programming [11], [12]. Similar to our work, these studies use a form of state abstraction to compute coarse policies before working out the details in fine-grained policies. However, the coarse policies are not valid for the whole state space and they are used only as initial approximations, and they are subsequently refined incrementally until the lowest-level policy is found; thus losing some of the advantages of high-level policies, such as reuse of previously computed policies.

Within reinforcement learning, which in many cases can be viewed as approximate, sampling-based dynamic programming techniques, there is a growing literature on exploiting hierarchy. Some of those studies [4], [5], [1] similarly allow reuse of previously computed value functions. However, all of these methods are model-free learning methods, which means they require many (often millions of) interactions with the environment before they learn a task, and they do not exploit the more powerful dynamic programming methods afforded by models. Furthermore, most of those methods do not exploit state abstraction which is very beneficial for hierarchical methods (but see [4], [1]), or they do not have the same performance guarantees as our method (but see [5]).

## IV. EXPERIMENT BASED ON ARTIFICIAL DATA

### A. The low-level map and clustering method

We first demonstrate our hierarchical method on artificial data, allowing us to investigate it under strictly controlled conditions. The data set, corresponding to the lowest level MDP  $\mathcal{M}^0$ , is generated as follows. First, 4000 points are randomly generated and placed in a simulated 2D 10 m by 10 m world. All points becomes states. Next, Euclidean distances are computed between all states. States are connected by possible transitions (edges) if and only if the Euclidean distance  $d(s, s')$  between them is less than 1 m, and the reward (cost) of the transition is  $r(s, s') = -d(s, s')$ . An action  $a'$  takes the system from a state  $s$  to the desired successor  $s'$  with a randomly generated probability  $0.5 < p(s'|s, a') < 1$ ; the remaining probability mass is distributed evenly over all other successors.

A 2-level hierarchy is constructed. To cluster the states of  $\mathcal{M}^0$  into higher level states for  $\mathcal{M}^1$ , we use the *normalized graph cut* algorithm from graph theory [7], explained in more detail for our particular type of application in [15]. Essentially, the algorithm cuts edges so as to arrive at unconnected subsets of the overall graph (MDP in our case), each of which becomes a cluster. It minimizes the number of edges it has to cut to yield the desired number of clusters, and

simultaneously maximizes the number of edges within each cluster. The normalized graph cut algorithm is appropriate for our problem for multiple reasons. First of all, it will often provide cluster boundaries at intuitive places, e.g. at narrow transitions (doors) between larger spaces (rooms). Secondly, if it fulfils its objective of cutting only one edge to distinguish between two clusters, our hierarchical method will lead to optimal paths between all states in the two clusters (because both hierarchical methods and flat methods need to pass through this one transition). Thirdly, by its very nature it enforces the Markov property for the higher level states. We use the normalized graph cut algorithm based on distances between states, with 20 clusters, and subsequently construct  $\mathcal{M}^1$  as described in section III-A.

### B. Results

We use the hierarchical value iteration method described in Algorithm 1 to compute policies to 100 randomly selected states. The results are compared to standard value iteration (section II-B), performed on the flat, low-level MDP  $\mathcal{M}^0$ .

Table I summarizes the results. Even with just one target location that must be planned for ( $\#ValUpd1st$ ), the number of value updates for the hierarchical method is significantly lower than for the standard flat method: 494, 652 vs. 3, 351, 160. This is due to the reduced state sets of the hierarchical method’s MDPs at both the higher and the lower level, leading to fewer states to update and faster convergence.

The difference in total required value updates ( $\#ValUpdTot$ ), and the difference in total run time ( $TotTime$ , on a standard P4 PC) quickly becomes very large as more target locations must be planned for, thanks in part to the hierarchical method’s frequent reuse of previously computed value functions ( $\#Reuse$ ). Importantly, the average time of computing a policy for a single target location ( $AvTime$ ), which is especially important for real-time control of a mobile robot, is only 0.5 seconds for our hierarchical method, as opposed to the 46.8 seconds of the standard, flat method. These numbers show that in large problems the hierarchical method remains feasible, especially for real-time applications, when the standard flat method is no longer feasible. However, the average maximum likelihood path length ( $AvPath$ ) for the hierarchical method is 5.61, which is higher than the standard flat method’s (optimal) value of 5.26.

## V. EXPERIMENT BASED ON REAL ROBOT DATA

### A. The low-level map and clustering method

Next, we turn to real robot data [15]. The data were obtained using our mobile robot equipped with an omnidirectional camera navigating in an office environment (our university building). The use of vision was dictated by the requirement to operate in realistic, natural environments. The data consist of 234 panoramic images taken at regularly spaced intervals in an office environment (see figure 3), which are assumed to have been obtained in an initial “exploration” phase. The observed environment essentially consists of 3

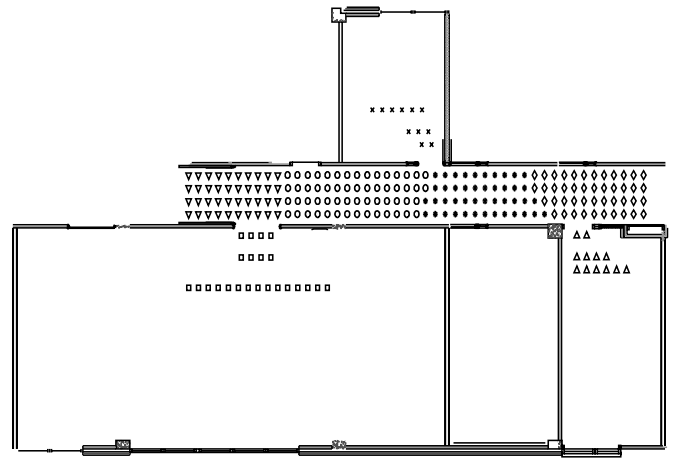


Fig. 3. Bird’s eye view of the office environment, with the locations where images were captured. Different symbols indicate different clusters, corresponding to different high-level states.

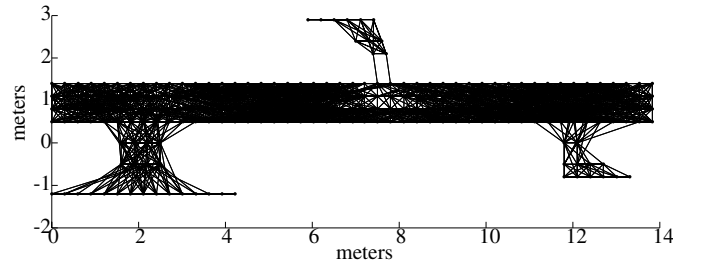


Fig. 4. The MDP ( $\mathcal{M}^0$ ) extracted from the images taken in the office environment.

rooms, connected through a single corridor. This database constitutes an appearance-based map, and each image becomes a state in  $\mathcal{M}^0$ .

Given the database of images, from each image a set of distinctive local image features is extracted, using the SIFT method [10]. SIFT features in different images are subsequently matched, and matches are possibly rejected due to geometric constraints (see [15] for details). If a sufficient number of matches remain, the images are sufficiently similar and are assumed to come from similar locations. In this case, it is possible to navigate from one state to the other, using homing based on matched features. Therefore, an edge (possible transition in the MDP) is added between the two states. In this experiment, we do not compute more accurate estimates of Euclidean distance or transition probability between two states, so we simply assume  $r(s, s') = -1$  and  $p(s'|s, a') = 1$  if an edge exists and 0 otherwise. Figure 4 shows the resulting MDP  $\mathcal{M}^0$  within the office environment.

As in the previous section,  $\mathcal{M}^0$  is clustered using the normalized graph cut algorithm, using 7 clusters. Figure 3 shows the clustering obtained in this case. Note how separate rooms become separate clusters. The corridor is also divided into a number of clusters.

### B. Results

We do a similar experiment as in the previous section. We use our hierarchical value iteration method to compute poli-

TABLE I

PLANNING RESULTS FOR BOTH DATA SETS (*Art.*: ARTIFICIAL DATA, *Real*: REAL ROBOT DATA). *DP* IS STANDARD, FLAT DYNAMIC PROGRAMMING, *HDP* IS HIERARCHICAL DYNAMIC PROGRAMMING. *#ValUpdTot* IS THE TOTAL NUMBER OF REQUIRED VALUE UPDATES. *#ValUpd1st* IS THE NUMBER OF VALUE UPDATES FOR THE FIRST TARGET LOCATION. *#Sweeps* IS THE TOTAL NUMBER OF SWEEPS THROUGH THE DATA SET. *TotTime* IS THE TOTAL RUN TIME (IN SECONDS). *AvTime* IS THE AVERAGE TIME OF COMPUTING A POLICY FOR A SINGLE TARGET LOCATION. *#Reuse* IS THE NUMBER OF TIMES A VALUE FUNCTION IS REUSED. *AvPath* IS THE AVERAGE MAXIMUM LIKELIHOOD PATH LENGTH TO THE TARGET.

Data	Method	#ValUpdTot	#ValUpd1st	#Sweeps	TotTime	AvTime	#Reuse	AvPath
Art.	DP	482,115,441	3,351,160	120,559	4,684 s	46.84 s	0	5.26
Art.	HDP	4,207,787	494,652	20,454	50 s	0.50 s	1,886	5.61
Real	DP	445,496	1,631	1,912	2 s	0.01 s	0	3.48
Real	HDP	36,328	807	895	1 s	0.003 s	1,622	4.13

cies to all possible states, comparing the results to standard, flat value iteration.

Table I summarizes the results. As in the previous section, it is apparent that the hierarchical method has clear computational advantages over the standard flat method. For the first target location that must be planned for, the number of value updates for the hierarchical method is 807 as opposed to the standard flat method's 1,631. To plan for all target locations, the number of value updates is 36,328 for the hierarchical method vs. 445,496 for the standard flat method. This experiment shows the viability of our hierarchical method on real robot data and its possible advantage over the standard, flat method, which will be especially relevant in larger real-time robot applications.

## VI. CONCLUSIONS

The results of this paper show the feasibility and promise of the hierarchical mapping and planning approach for robot navigation. Robot environments were formalized as a hierarchy of MDPs and subsequently solved using a variation of dynamic programming. Our hierarchical dynamic programming approach leads to significant savings in terms of the number of value updates required for convergence, compared to standard, flat dynamic programming, especially when the state space becomes large and when navigation policies must be computed to many target locations. This advantage is due to reduced state spaces and efficient reuse of previously computed value functions. This may come at the cost of somewhat longer paths to target locations, but given that the Markov property holds for the low-level and high-level MDPs, value functions and policies are computed which always correspond to possible paths to the goal and which are optimal given the hierarchy.

Future work includes testing the algorithms in a real robot experiment (as opposed to the simulation experiment based on real robot data used now) and investigating hierarchies with levels  $N > 2$  for very large, realistic tasks. Furthermore, the hierarchical method could be adapted for tasks other than path planning, including infinite horizon discounted reward problems. It would also be interesting to investigate ways to reduce the hierarchical method's disadvantages, e.g. by adapting the clustering method to make longer than optimal paths

unlikely, by allowing refinements of computed policies when time permits, and by making estimates of higher-level state transition probabilities and rewards more accurate. Finally, frameworks based on MDPs which add partial observability (POMDPs) or multiple agents can similarly benefit from hierarchical methods.

## REFERENCES

- [1] B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In *Proceedings of the 8-th Conference on Intelligent Autonomous Systems, IAS-8*, pages 438–445, 2004.
- [2] D. P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, Belmont, MA, 1995.
- [3] J. M. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. E. Schneider, J. Strikos, and S. Thrun. The mobile robot RHINO. *AI Magazine*, 16(2):31–38, 1995.
- [4] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, 1993.
- [5] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
- [6] C. Galindo, J.-A. Fernandez-Madrigo, and J. Gonzalez. Improving efficiency in mobile robot task planning through world abstraction. *IEEE Transaction on Robotics*, 20(4):677–690, 2004.
- [7] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–904, 2000.
- [8] S. Kambhampati and L. S. Davis. Multiresolution path planning for mobile robots. *IEEE Journal of Robotics and Automation*, 2(3):135–145, 1986.
- [9] B. J. Kuipers. Representing knowledge of large-scale space. Technical Report TR-418 (revised version of Doctoral thesis), MIT Artificial Intelligence Laboratory, July 1977.
- [10] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [11] J. K. Peterson. Path planning in analog valued obstacle arrays using hierarchical dynamic programming and neural networks. In *Proceedings of the Artificial Neural Networks in Engineering (ANNIE91) Conference*, pages 789–794, 1991.
- [12] C. Raphael. Coarse-to-fine dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1379–1390, 2001.
- [13] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.
- [14] N. Tomatis, I. Nourbakhsh, and R. Siegwart. Simultaneous localization and map building: A global topological model with local metric maps. In *Proceedings of IROS*, 2001.
- [15] Z. Zivkovic, B. Bakker, and B. Kröse. Hierarchical map building using visual landmarks and geometric constraints. In *Proceedings of IROS*, 2005.