To appear in: *Proceedings of the IEEE International Conference on Neural Networks*
San Francisco, CA, March 28-April 1, 1993

Exploration and Model Building in Mobile Robot Domains

Sebastian B. Thrun [1]
University of Bonn, Institut für Informatik III
Römerstr. 164, D-5300 Bonn 1, Germany
E-mail: thrun@uran.informatik.uni-bonn.de

Abstract— **I present first results on COLUMBUS, an autonomous mobile robot. COLUMBUS operates in initially unknown, structured environments. Its task is to explore and model the environment efficiently while avoiding collisions with obstacles. COLUMBUS uses an instance-based learning technique for modeling its environment. Real-world experiences are generalized via two artificial neural networks that encode the characteristics of the robot's sensors, as well as the characteristics of typical environments the robot is assumed to face. Once trained, these networks allow for the transfer of knowledge across different environments the robot will face over its lifetime. COLUMBUS' models represent both the expected reward and the confidence in these expectations. Exploration is achieved by navigating to low confidence regions. An efficient dynamic programming method is employed in background to find minimal-cost paths that, executed by the robot, maximize exploration. COLUMBUS operates in real-time. It has been operating successfully in an office building environment for periods up to hours.**

I. Introduction

I report first results on robot exploration in mobile robot domains. In contrast to many other approaches to robot learning, I consider a robot that has no other task than maximizing its knowledge about the initially unknown environment, while avoiding negative reward. The robot at hand, COLUMBUS (Figure 1), is a wheeled mobile robot equipped with a proximity sensor, a sonar sensor that allows to sense distances to objects next to the robot by emitting sonar signals, and a motion sensor for monitoring the motion of the wheels and detecting collisions. COLUMBUS operates in environments such as office buildings or private homes and has to avoid negative reward that will be received when colliding with obstacles. It operates in real-time.

COLUMBUS employs a local, instance-based learning technique to model its environment. Rather than fitting a global monolithic model, experiences are remembered explicitly, and functions are approximated locally with the help of artificial neural networks. Many global approaches to function fitting (e.g., with a single monolithic neural network) in mobile robot domains either fail in complex environments due to effects like un-/relearning, or demand many well-distributed training examples and have often been tested in simplified simulations only [2, 9, 15, 18]. Modularized, local and instance-based approaches to function approximation have often been reported to generalize better from fewer or ill-distributed examples [1, 5, 6, 7, 10, 12]. Instance-based curve fitting techniques have also been applied successfully to fairly complex robot control learning problems, including the work by Atkeson and Moore. In COLUMBUS' approach to model building, experiences are remembered explicitly. They are generalized via two artificial neural networks, one for *sensor interpretation* and one for *confidence assessment*. These networks encode the specific characteristics of the sensors as well as those of typical environments of a mobile robot, thus capturing knowledge independent of any particular environment the robot might face. Once trained, they provide an efficient way of knowledge transfer from previously explored environments to new environments. For example, based on knowledge acquired in previously explored environments COLUMBUS was found to avoid collisions almost completely in new environments. In contrast, tabula rasa learning methods will cause collisions in any new, unknown environment, before the robot eventually learns to avoid them.

COLUMBUS top-level goal is efficient exploration. The approach taken in this paper is motivated by earlier research on exploration in the context of reinforcement learning [18]. Theoretical results on the efficiency of exploration indicate the importance of the exploration strategy for the amount of knowledge gained, and for the efficiency of learning control in general. It has been shown that for certain hard deterministic environments, that an autonomous robot can face, exploration strategies such as random walk result in an expected learning time that scales at least exponentially with the number of states the environment can take [19]. In contrast, more thoughtful exploration techniques, such as *"go to the least explored location,"* have been shown to reduce the complexity to a small polynomial function in the size of the state space [8, 17]. While these results may be theoretically significant, their relevance and implications for practical research in robot exploration are unclear. This is because the best known worst-case bounds for the complexity of exploration are still too large to be of any practical meaning, given the complexity of environments and state spaces faced by

[1] This research was done while the author was with Carnegie Mellon University, School of Computer Science, Pittsburgh PA 15213.

This research was sponsored in part by Siemens Corporation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Siemens Corp.

Figure 1: (a) COLUMBUS is a wheeled HERO-2000 robot with a manipulator and a gripper. It is equipped with a sonar sensor on the top of the robot that can be directed by a rotating mirror to give a full 360° sweep (24 values). Sonar sensors return approximate echo distances. Such sensors are inexpensive but very noisy. (b) COLUMBUS explores an unknown environment. Note the obstacle in the middle of the laboratory. Our lab causes many malicious sonar values, and is a hard testbed for sonar-based navigation. For example, some of the chairs absorb sound almost completely, and are thus hard to detect by sonar.

a robot acting in the real world. Furthermore, the theoretical results ignore the ability of mobile robots to gain knowledge by sensing its environment, which allows the robot to make predictions for neighboring locations. Such predictions, of course, may reduce the number of exploration steps drastically. However, the intuition behind the theoretical results carries over to mobile robot domains. An efficiently exploring robot should not explore by selecting actions randomly. Instead, it should actively maneuver to poorly explored parts of the environment, in order to maximize knowledge gain. COLUMBUS' exploration strategy follows this principle. Its model keeps track of how well its environment is explored, and based on this knowledge the robot is maneuvered to poorly explored regions.

II. Modeling the Environment

In this section, the instance-based model building technique developed for COLUMBUS is described. Let x_i denote the *location* (position, state)[2] of the mobile robot at time step i, s_i denote the vector of *sensations* (sonar measurements in the case of COLUMBUS) at this time, and r_i the (collision) reward received at this time. Generally speaking, an *adaptive model* $\mathcal{M}$ is a function that generalizes from a finite set of examples (data points) $\{\langle x_i, s_i, r_i\rangle | i = 1...n\}$ to arbitrary new positions x in the domain:

$$r \quad = \quad \mathcal{M}(\{\langle x_i, s_i, r_i\rangle | i = 1 \ldots n\}, x)$$

In our case, r denotes the expected reward received when entering location x.

Different model identification procedures differ in the way in which they combine and interpolate between points (inductive bias). The approach taken in COLUMBUS is to split the function $\mathcal{M}$ into n pieces, one for each data point. More specifically, $\mathcal{M}$ is realized using a pair of artificial neural networks: the *sensor interpretation network* $\mathcal{R}$ and the *confidence estimation network* $\mathcal{C}$. These networks are trained using the backpropagation training procedure [13] to encode the specific characteristics of the sensors as well as those of typical environments of a mobile robot.

More specifically, the **sensor interpretation network**

$$r_i \quad = \quad \mathcal{R}(s_i, x{-}x_i) \quad = \quad \mathcal{R}(s_i, \Delta x)$$

(c.f. Fig. 2a) maps a *single* experience (x_i, s_i) to an estimation of reward at x. Here Δx is the difference between x_i and x, making $\mathcal{R}$ independent of absolute coordinate values. Fig. 3a illustrates the output of $\mathcal{R}$ in the mobile robot domain. Here the distance measurements from the sonar sensor are mapped to expected reward (generalized occupancy). $\mathcal{R}$ is trained in a supervised manner: The robot operates and takes sensor measurements in a *known* environment where regions which the robot can physically not enter are labeled with negative reward.[3]

Instead of *one* estimate of expected reward $r = \mathcal{M}(x)$ at location x, $\mathcal{R}$ returns n estimates r_i, one for each data point (s_i, x_i). There are several reasons why combining multiple interpretations r_i is necessary. First, many sensors such as sonars are noisy, and often fail to sense the environment accurately. Second, $\mathcal{R}$ models an inherently unpredictable function. For example, a sonar sensor cannot "look through a wall." Thus if the query point x and x_i are separated by an obstacle, r_i will usually be wrong. In what follows, a scheme for combining multiple interpretations based on estimating confidence is described.

The **confidence network** (Fig. 2b)

$$c_i \quad = \quad \mathcal{C}(s_i, \Delta x)$$

maps a single experience (x_i, s_i) to a scalar in $(0, 1)$, again with

[2]Throughout this section it will be assumed that there is an accurate way to estimate the location x_i of the robot in some global coordinate frame. This assumption is relaxed in section IV. , where a technique for re-estimating the location of the robot based on sensations is described.

[3]Note that in the current implementation, this is done with a simulator that models sonar sensors as well as sensor noise. Simulation is significantly cheaper and faster than real-world experimentation, and I empirically found $\mathcal{R}$ to be accurate enough even if trained on simulated data. All results displayed in figures, however, are generated from real-world data.

In general, training these networks can be interleaved with exploration, allowing for cross-environment transfer of learned knowledge for model identification.

(b)

Figure 2: (a) sensor interpretation network $\mathcal{R}$ and (b) confidence network $\mathcal{C}$.

$\Delta x = x - x_i$. Note that the interpretation network $\mathcal{R}$ will never generalize perfectly, i.e., its error will never approach zero. This effect is utilized for training the confidence network. Using an independent test set for $\mathcal{R}$ (after training), $\mathcal{C}$ is trained to estimate the *expected error* $|\mathcal{R}(s_i, \Delta x) - reward(x)|$ of $\mathcal{R}$ (normalized to $[0, 1]$). Since $\mathcal{C}$ estimates the expected error of r_i, the confidence in r_i is low if c_i is large and vice versa. It is straightforward to use this error estimate in the integration of the r_i by weighting interpretations r_i with their *confidences*, defined as $-\ln(c_i)$.

Figure 3b illustrates confidence estimations, corresponding to the reward estimations shown in Figure 3a. Note that predictions of freespace and walls usually have high confidence, whereas for example the confidence in predictions behind walls is low. As can be seen from some of the examples in this figure, the confidence network deals with noise by assigning low confidence to potentially noisy sensations.

Finally, reward estimates r_i are combined according to their confidence estimates $-\ln c_i$, using the formula.

$$\mathcal{M}(x) = \frac{\sum_i -\ln c_i(x) \cdot r_i(x)}{c_{\mathcal{M}}(x)} \quad \text{with} \quad c_{\mathcal{M}}(x) = \sum_i -\ln c_i(x)$$

The term $c_{\mathcal{M}}(x)$ is called the *cumulative confidence* at x. Knowing that COLUMBUS' sonar sensors have a maximum range of perception (10.5 feet), $i \in \{1 \ldots n\}$ sums only over those data points that are close enough to x, i.e., for which $|x - x_i|$ is smaller than the maximal sonar perception range. In Figure 4b, a compiled model after 27 exploration steps is shown.

III. Anytime Planning for Exploration in Real-Time

Exploring and modeling initially unknown environments is the top-level goal of COLUMBUS. In order to find low-cost paths to the unexplored, the model is discretized yielding a grid representation of the environment, and dynamic programming is employed to propagate exploration utility through this discretized model [3, 16]. More specifically, this is done in the following way: To each grid point x in the discretized model there is a real-valued *exploration utility* $U(x)$ associated. Initially, the exploration utility of x is set to the negative cumulative confidence $-c_{\mathcal{M}}(x)$. Cumulative confidence (c.f. Equation (1)) is a straightforward measure to estimate the utility of exploring a location: The lower $c_{\mathcal{M}}(x)$, the less explored is x and the higher is the (initial) exploration utility of x, and vice versa. All grid points are then iteratively updated according to the maximum

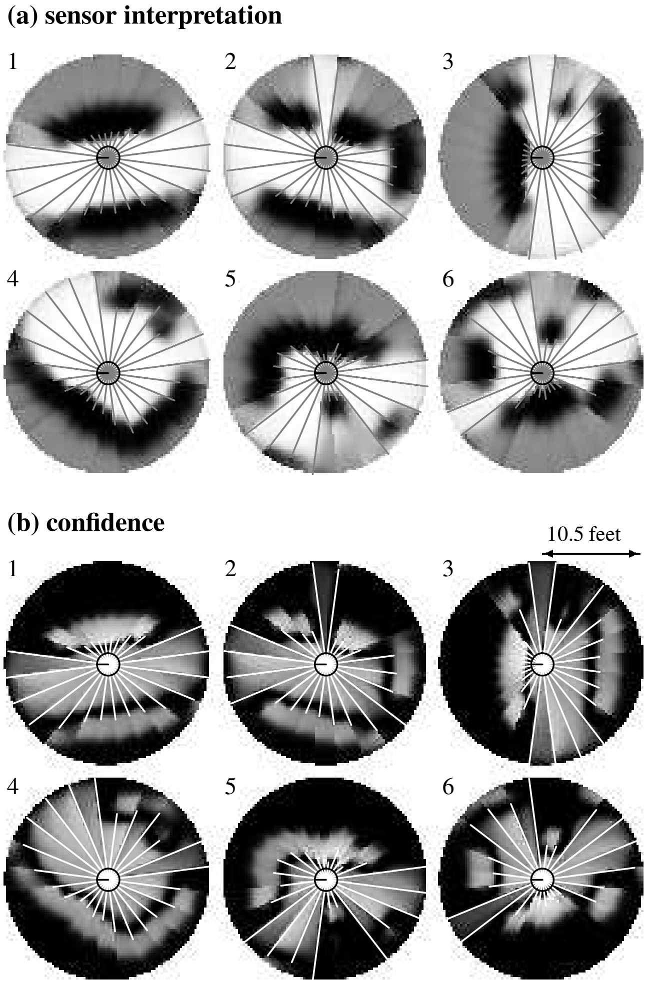

Figure 3: (a) Examples of sensor interpretation using the $\mathcal{R}$ network. Lines indicate sonar measurements (distances), and the region darkness represents the expected collision reward for surrounding areas (dark values indicate negative reward). Examples are: 1. hallway, 2. hallway with open door. 3. hallway with human walking by, 4. corner of a room, 5. corner with obstacle, 6. several obstacles. (b) The corresponding confidences C in the interpretations shown in a. The darker the color, the larger the expected error and the lower the confidence. Low confidence regions include the boundary region between freespace and obstacles, as well as regions beyond obstacles.

exploration utility of their neighbors y of the grid:

$$U(x) \quad \longleftarrow \quad -L(x) \; + \; \max_y \; U(y)$$

Here the expected (negative) reward by the model functions as *costs*[4] $L(x) > 0$ of moving from x to y, and is used for weighting the flow of exploration utilities. This implies that exploration utilities are predominantly propagated through freespace, while places with negative predicted reward such as walls and obstacles block the flow of utility. The resulting exploration utilities represent anytime plans for arbitrary robot locations, given that the robot applies steepest ascent therein. Figure 4 displays a model and the corresponding exploration utility landscape after 27 steps of exploration.

[4]The actually implemented cost function is a complex monotonically decreasing function of the expected reward.

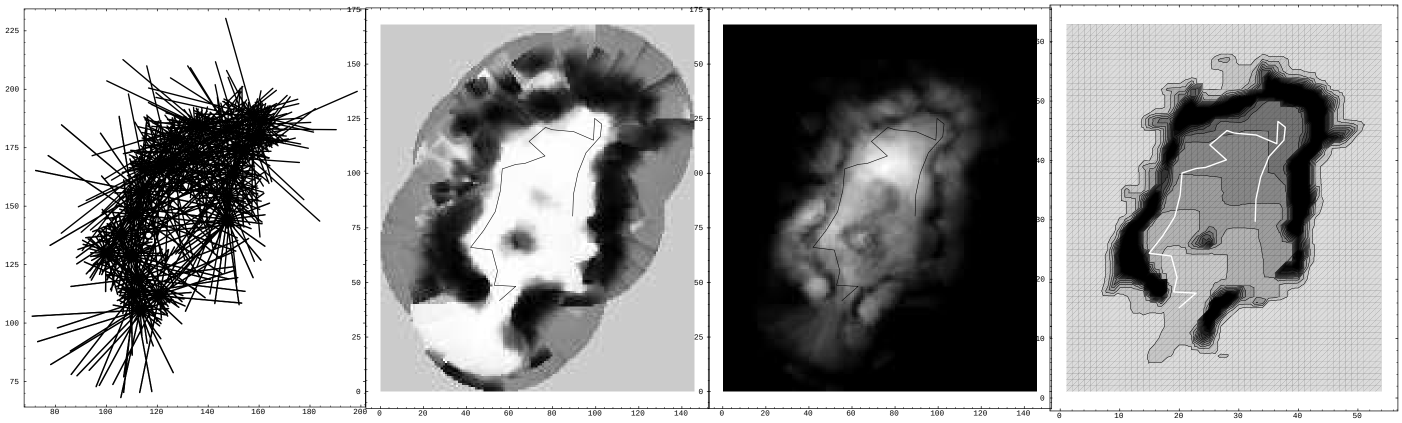

Figure 4: **Integration of several measurements along an exploration path:** (a) Raw, noisy sensor input on an exploration path with 27 measurements. (b) Resulting model, corresponding to the lab shown in Figure 1b (lab doorway is toward bottom left). The path of the robot is also plotted (from right to left), demonstrating the exploration of the initially unknown lab. (c) Cumulative confidence. (d) Exploration utilities. By local gradient descent in exploration utility space (i.e., moving to the brightest reachable location), the robot moves to the closest unexplored region on a minimal-cost path. In the next steps, the robot will pass the door of the lab and explore the hallway.

Because COLUMBUS shall explore in real-time, actions are immediately generated once the previous action execution is finished, regardless whether the propagation of exploration utility has converged or not. In principle this may lead to non-optimal actions. However, since the exploration values are propagated not only to the current location of the robot, but to all points in the discretized model, they can be reused for later planning once the robot location has been changed. Reusing plans was observed to result in optimal plans after surprisingly short planning times, although plain dynamic programming is known to be slow in complex domains.[5]

Since COLUMBUS' model of the environment is continuously updated during exploration according to new sensor information, dynamic programming has to be modified to deal efficiently with the resulting changes in the cost function and the cumulative confidence. In general, dynamic programming may cause long planning time if the model changes during planning. More specifically, the problem that may occur when changing the cost function or the cumulative confidence is the *overestimation problem* for exploration utilities: Assume the planner assigns high utility to two adjacent grid points that, after updating the model based on new sensor information, will now have low utility. Dynamic programming may take many iterations before converging to the lower utilities. This is because each of the large exploration utilities will be updated with a large value, justified by the observation that there is an adjacent grid point with a large utility. Only the costs of moving from one grid point to another can cause the utilities to decay over time. This effect was frequently observed to cause very long planning durations. In order to overcome the overestimation problem, COLUMBUS' planner works in three phases:

1. If a new sensation is received, the grid representation of both the model and the confidence map is updated according to this sensor information, using the networks $\mathcal{R}$ and $\mathcal{C}$.
2. Each overestimated utility, i.e., each utility that would have been set to a smaller value by the next dynamic programming step, is set to $-\infty$. This step is repeated until no overestimated utility can be identified, resulting in a strictly underestimating set of exploration utilities.
3. Finally, dynamic programming is applied to propagate exploration utilities as discussed above.

Whenever the robot finishes the execution of an action, a simple search procedure that maximizes exploration utility is employed to determine the next action, using the current estimate of exploration utilities. This ensures that COLUMBUS operates in real-time: Even if planning is not completed, actions are generated and executed. It should be noted that this planner can also be, and in fact is in the current implementation, used for planning paths to goal locations specified by the user.

IV. Experimental Design and Results

In Section II. it was assumed that the location of the robot can always be determined accurately. However, real-world robots usually suffer from cumulative control errors caused by inaccurate effectors. After some time of operation, location estimates based on dead-reckoning are usually significantly wrong. COLUMBUS uses the networks $\mathcal{R}$ and $\mathcal{C}$ also for relocating itself on-the-fly with respect to its model. This is done by building a *local model* (interpretation) from the most recent sensation, s_{n+1}, and then maximizing the match between this local model and the current global model obtained from the previous n sensations $\{s_1, s_2, \ldots, s_n\}$. In COLUMBUS, gradient descent search is used to estimate the location of the robot x_{n+1}. More specifically, the squared deviation of the global and the local model (weighted by prediction confidences) is measured over a set of randomly drawn nearby sample points. Online position control is achieved by minimizing this match error. Since neural networks represent differentiable functions, these error gradients can be propagated through the sensor interpretation network, resulting

[5] The planning speed benefits also from the fact that the internal representation of the environment is only two-dimensional, neglecting COLUMBUS' rotational degree of freedom.

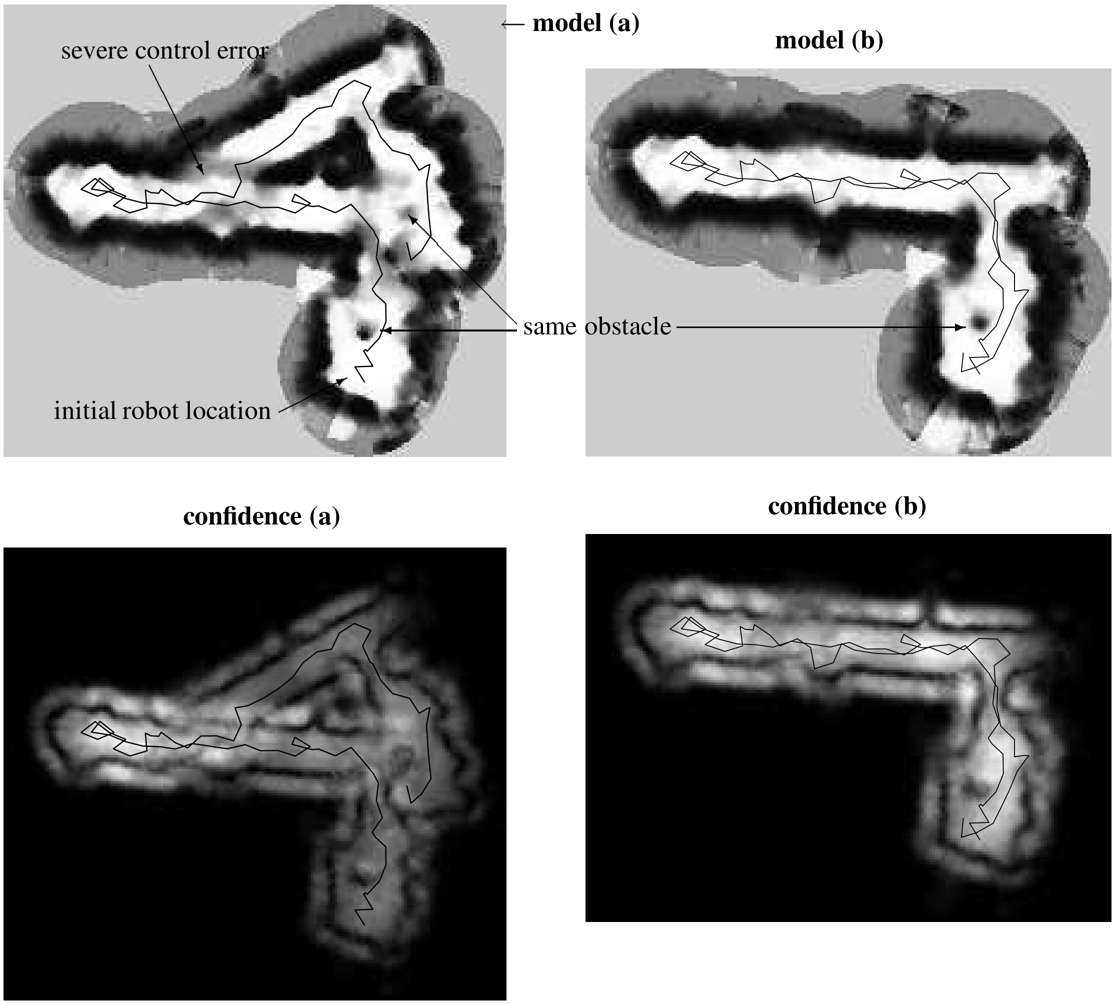

Figure 5: **Online position control:** (a) In this experiment, I invoked one severe control error by rotating the robot manually, in addition to the normal cumulative control errors. Without correction, the resulting model is wrong and therefore useless. (b) By maximizing the match between the global model and the interpretation of the most recent sonar values (local model), control errors can be successfully identified and corrected. Note that in this experiment, the robot was told to return to the lab after some exploration.

in the derivatives of the match error with respect to the input values of these networks. These gradients are used for gradient descent search in the location space of the robot. By iteratively adjusting the internal belief of the location of the robot, the match error is minimized. This online position control procedure was found to work successfully even in the presence of severe errors in dead-reckoning. An example is shown in Figure 5.

Both the sensor interpretation function $\mathcal{R}$ and the confidence estimation function $\mathcal{C}$ were represented by backpropagation networks with one hidden layer and eight hidden units. Instead of processing all 24 sonar values by these networks, only the four to a query point x nearest sonar values were provided as an input. By ignoring most of the sensor information, the processing time was significantly reduced when updating the discretized versions of the model, as was the number of training examples required for training $\mathcal{R}$ and $\mathcal{C}$. However, omitting sensor values is only valid, if these are independent of the expected reward at point x. This is approximately the case for COLUMBUS. Additional speedup was obtained by ignoring COLUMBUS' rotational degree of freedom, as mentioned above. Henceforth the dimensionality of the model was two, and the cost of rotations was considered to be 0. Introducing a third dimension would have had a negative impact on memory requirements and planning times in COLUMBUS. The grid size in the planner was 4 inches. COLUMBUS planner can deal with large areas (500 000 square feet and more), limited only by the memory capacity of the workstation used for planning.

COLUMBUS actual implementation is modularized and distributed. Modules (map builder, planner, position controller, central controller and graphical user interface) are connected using the Task Control Architecture [14], which allows to execute the programs on several SUN SPARC workstations in parallel. Robot actions take usually between 3 and 12 seconds, plus approximately 3 seconds for transmitting sensor and control information by a radio link. During extensive experimentation I observed roughly the following timing behavior: 1.5 sec for updating the discretized version of the model, 2 to 20 sec for complete planning from scratch, and between 0 and 4 for planning when reusing earlier plans (planning took up to 120 sec without the utility reset mechanism described in the previous section). The position control mechanism described above did run endlessly in the background, interrupted only by new sensor information.

V. Discussion

COLUMBUS is an autonomous mobile robot, whose goal it is to explore and model unknown environments efficiently. It employs an instance-based approximation technique based on neural networks for modeling the environment, and an anytime planner based on dynamic programming for planning low-cost paths to poorly explored areas. COLUMBUS has been successfully operated in the hallways and labs at CMU for periods up to hours.[6] In order to do so in dynamic environments such as a office buildings, the current implementation features a fast obstacle detection and avoidance mechanism that is not described here. However, COLUMBUS' models assumes static environments. It seems to be feasible to extend this approach to slowly changing environments (e.g., by decaying confidence over time).

In principle, the artificial neural networks $\mathcal{R}$ and $\mathcal{C}$ employed for model building and exploration allow for knowledge transfer and thus a synergy effect across multiple environments. This is because these network represent robot-specific knowledge that is independent of particular environments at hand. However, in the current real-world implementation this synergy effect has been only partially demonstrated, since the networks are pretrained in simulation. One of the main limitations of this approach is that these networks have to be trained in a supervised manner, requiring a *known* environment. Although in principle COLUMBUS is able to label its environment autonomously by sensing collisions directly (using the motion sensors of its wheels), I preferred simulation, since it facilitates supervised learning, and since real-world collisions will ultimately damage the robot.

In order to operate in real-time, COLUMBUS features an anytime planner based on dynamic programming. I have empirically found that this planner generates appropriate actions in real-time, even though the dynamic programming process might not have fully converged. This planner, however, operates on a two-dimensional representation of the environment. In order to scale up to higher dimensional environments the approach taken has to be modified.

Future research will also include comparisons to other approaches to model building and path planning, for example [4, 11].

Acknowledgment

I thank Tom Mitchell and Alex Waibel for invaluable advice, and Long-Ji Lin, Hans Moravec, and the ODYSSEUS team at CMU, namely Reid Simmons, Charalambos Athanassiou, John Cheng, Lonnie Chrisman, Richard Goodwin, Goang-Tay Hsu, and Hank Wan, for being an excellent team as well as for many discussions. Furthermore, I thank Reid Simmons and Chris Fedor for providing the Task Control Architecture software

[6] In fact, COLUMBUS' approach to model building and position control has been successfully used as part of the CMU entry "ODYSSEUS" in the first AAAI robot competition in California, 1992.

References

[1] Christopher A. Atkeson. Using locally weighted regression for robot learning. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 958–962, Sacramento, CA, April 1991.

[2] Jonathan R. Bachrach. A connectionist learning control architecture for navigation. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 457–463, San Mateo, 1991. Morgan Kaufmann.

[3] Andy G. Barto, Steven J. Bradtke, and Satinder P. Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report COINS 91-57, Department of Computer Science, University of Massachusetts, MA, August 1991.

[4] Alberto Elves. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

[5] Dieter Fox, Volker Heinze, Knut Möller, Sebastian B. Thrun, and Gerd Veenker. Learning by error-driven decomposition. In Simula, Kohonen, editors, *Proceedings of International Conference on Artificial Neural Networks*, Amsterdam, 1991. Elsevier Publisher.

[6] Jerome H. Friedmann. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, March 1991.

[7] Robert A. Jacobs and Michael I. Jordan. A modular connectionist architecture for learning piecewise control strategies. In *Proceedings of the American Control Conference*. Dept. of Brain and Cognitive Sciences, MIT, 1991.

[8] Sven Koenig. The complexity of real-time search. Technical Report CMU-CS-92-145, Carnegie Mellon University, April 1992.

[9] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of AAAI-91*, Menlo Park, CA, July 1991. AAAI Press / The MIT Press.

[10] Andrew W. Moore. *Efficient Memory-based Learning for Robot Control*. PhD thesis, Trinity Hall, University of Cambridge, England, 1990.

[11] Hans P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.

[12] Steven J. Nowlan. Competing experts: An experimental investigation of associative mixture models. Technical Report CRG-TR-90-5, Dept. of Computer Science, University of Toronto, Canada, September 1990.

[13] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II*. MIT Press, 1986.

[14] Reid Simmons. Concurrent planning and execution for autonomous robots. *IEEE Control Systems*, 12(1):46–50, February 1992.

[15] Satinder P. Singh. The efficient learning of multiple task sequences. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 251–258, San Mateo, CA, 1992. Morgan Kaufmann.

[16] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning, June 1990*, pages 216–224, 1990.

[17] Sebastian B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA 15213, January 1992.

[18] Sebastian B. Thrun. The role of exploration in learning control. In David A. White and Donald A. Sofge, editors, *Handbook of intelligent control: neural, fuzzy and adaptive approaches*, Florence, Kentucky 41022, 1992. Van Nostrand Reinhold.

[19] Steven D. Whitehead. A study of cooperative mechanisms for faster reinforcement learning. Technical Report 365, University of Rochester, Computer Science Department, Rochester, NY, March 1991.