Kikuo Fujimura*

Path Planning with Multiple Objectives

Most path planners are designed to generate a single path that is optimal in terms of some criterion such as path length or travel time. However, for realistic terrain navigation we wish to find a path that is reasonable to execute in a given environment. Therefore we must consider several factors, such as safety, time, and energy consumption. In this article the authors investigate how to find a set of paths (as opposed to a single path) so as to permit various choices concerning multiple criteria. They present simulation results to demonstrate the feasibility of the approach and discuss an extension to navigation in time-varying scenes.

This paper addresses the problem of finding a navigation path for a mobile robot that must consider multiple costs in navigation, such as the length of the path, traversability of the area, time to travel to the destination location, etc. In most path planning problems, we are interested in finding a path that is optimal in some single criterion such as path length. Practically, however, most navigation tasks must take into accounts other criteria at the same time. For example, we do not want a shortest path at the cost of safety along the path. This paper presents a method for generating a group of paths based on a given set of criteria so that the user can pick the one that best suits his objectives.

To illustrate the motivation for our study for multiobjective search, let us give one example from path search in a time-varying domain. In a time-varying environment, the minimum-length path and minimum-delay path are usually not the same. Delay is the time required to travel from start to goal points, and length is measured by the distance the robot has actually moved along the path. See Fig. 1 for an example. The environment contains an obstacle moving from left to right (Fig. 1(i)). Along the fastest path, robot motion would be via a corner of the moving obstacle (Fig. 1(ii)), while along the shortest path, the robot would wait until the obstacle is gone and follow a straight-line path (Fig. 1(iii)). Time-minimum paths are desirable for robots that need to arrive at the destination location as soon as possible, while robots may save energy along shortest distance paths. Between the two extremes, there are a variety of paths which are of practical value if they are sufficiently fast and short, although they may be neither of minimum-time nor minimum-length.

Practically, these are usually the paths that are desirable. We don't want an optimal path in terms of some single criterion at the cost of other equally important criteria. This is exactly the type of problem that our multiobjective search is designed for.

One may wonder why a conventional single objective path planner cannot be used

*Department of Computer and Information Science, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210. E-mail: fujimura@cis.ohio-state.edu Phone: 614-292-6730 Fax: 614-292-2911

1070-9932/96/$5.00©1996 IEEE

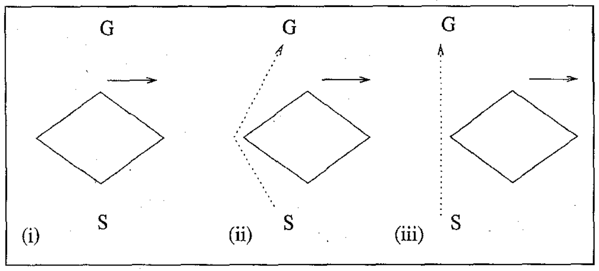


Figure 1. Fast path (ii) and short path (iii)

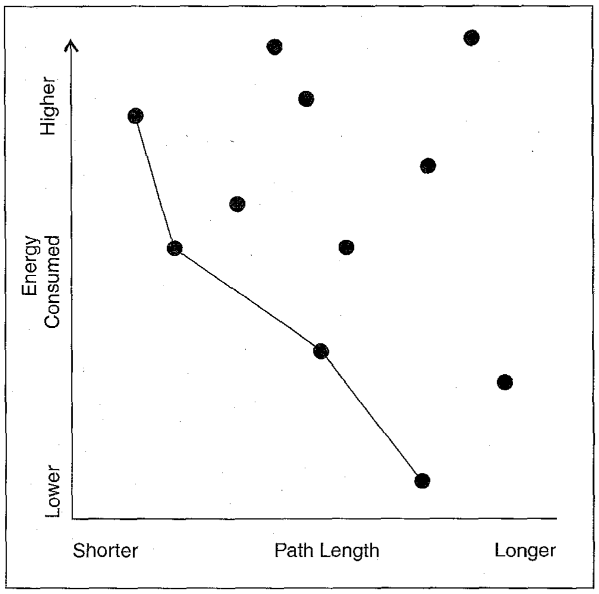


Figure 2. Nondenominated path

with its objective set as a weighted sum of multiple objectives, that is, cost = $\omega_1 c_1 + \omega_2 c_2 + \ldots$, where c_i is the i-th cost and ω is the weight for the i-th cost. By selecting appropriate weights, a path of desirable property may be obtained by a search with a single objective. For several reasons, multiple objective search may be better than a single objective search with weighted objectives. First, if the user wants to find the best paths in terms of a variety of objectives, search with a single objective may have to be run many times, whereas one execution of multiobjective search suffices to answer such queries. Secondly, a single search with weighted objectives cannot give a satisfactory answer to a query such as "Among the paths whose path-lengths are within c and safety factor is less than d, what is the path whose travel time is minimum?" whereas multiobjective search can. Finally, when some objectives are dependent on others, multiple objective search must be used to generate paths to generate an optimal solution. This point will be explained in more detail later.

We will first introduce search methods with multiple objectives and present our simulation results produced by our method. We will then compare our method with other approaches and discuss an extension to handle time-varying environments.

PATH SEARCH METHODS

In this section we present our multiobjective search algorithm in the context of static environments. Throughout this section, we use two criteria for path evaluation, energy consumption and path length, to illustrate our search method. In the simulation described in the next section, we measure energy consumption as a function of slope of the path being traversed, while path length is measured by the Manhattan distance. For a nonflat terrain, a shortest path may not necessarily be the path with the lowest energy consumption due to the presence of steep hills, etc. Suppose we have a set of paths $\pi_1, \pi_2, \ldots$, from the start point to a certain location in the map. Then, the paths can be plotted on a 2-dimensional chart as in Fig 2, where the horizontal axis represents path length and the vertical axis represents energy consumed. Each black dot represents a path. For our purpose, it is not necessary to keep all the black dots in the chart since some paths may be dominated by others. A path π_1 is said to dominate another path π_2 if path π_1 is better than π_2 in all the criteria considered. If this is the case, there is no point in keeping path π_2 for further processing. If no path dominates path π then π is said to be nondominated.

It is easy to confirm that a dot (representing a certain path) is nondominated when the lower-left quadrant centered at the dot does not contain any other dot. In Fig. 2, the points connected by the solid lines are nondominated. In the case of two objectives, the set of nondominated paths can be linearly ordered as $\pi_1, \pi_2, \ldots, \pi_n$. They satisfy the following nondominated property (NDP).

NDP

1. Path π_i is more economical (energy consumption is lower) than π_{i+1}.
2. Path π_{i+1} is shorter than π_i.
3. Path π_i is the most economical among paths whose path lengths are equal to or shorter than π_i.
4. Path π_i is the shortest among paths whose energy consumption is equal to or earlier than that of π_i.
5. The list is maximal to satisfy the above conditions.

This idea plays an important role in multiobjective search [12]. It should be clear that with this ordering it is relatively simple to answer queries such as

1. Find a path that is shorter than distance a and whose economy does not exceed b.
2. What is the shortest path among paths whose economy is less than c?

For n objectives, paths can be plotted in an n-dimensional space and nondominated paths are defined in a similar manner, although paths can no longer be ordered linearly. The goal is to find a set of nondominated paths at the destination point so that queries concerning multiple objectives can be answered.

Search Algorithm

Now we present our search algorithm for path planning with multiple objectives. For the sake of clarity, we present our method using only two objectives. Generalization to multiple objectives should be straightforward. We assume that the terrain is represented by a grid (pixel array) as is usually done for terrain navigation. We associate PATH-LIST to each pixel P_i to retain a list of nondominated paths from S to P_i where S is the start point. For two objectives, energy consumption and path length, an element of P_i 's PATH-LIST is a path of the form [*en, len, predecessor*], where *en, len,* and *predecessor* are the energy consumed so far to reach P_i, path length from S to P_i, and P_i 's predecessor along the path. Initially, PATH-LIST of all nodes in the grid are empty.

To calculate weights between pixel P and one of its neighboring pixels Q, we use procedure MOVE(P, Q), whose output is of the form [P, Q, *len*], where *en* and *len* respectively denote the accumulated energy consumption and path length from S to Q via P. We use a queue to store paths whose *en* and *len* are tentatively assigned. An element of the queue is of the same form as outputs of MOVE. The following algorithm *Find-Path* will create a set of nondominated paths for all pixels starting from a start pixel S.

Procedure Find-Path

1. Insert the start tuple [*nil*, S,0, 0] into the queue where t_1 is the start time.
2. If the queue is empty, then exit the procedure. Otherwise, pop an element, say [K_i, P_j, e, l], from the queue whose associated energy e is the lowest.
3. If l of the popped tuple in Step 2 is smaller than *len* of the last element of P_j's PATH-LIST, then enter [e,l,K_i] into P_j's PATH-LIST and enter output of MOVE(P_j,Q) into the queue for all Q adjacent to P_j in the grid. Repeat Steps 2 and 3.

In Step 3, we check if the popped element in Step 2, [e,l,K_i], may be added to PATH-LIST without violating the NDP. If it preserves the NDP, it is entered in PATH-LIST. At this point this element becomes a permanent element of P_j's PATH-LIST.

For example, suppose that pixel P has paths $\pi_1,\pi_2,...\pi_4$ satisfying the NDP, where π_1 is the lowest in energy consumption as in Fig. 2. Since elements with lower energy are popped first from the queue, a new path to pixel N, say π_5, cannot be lower in energy than π_4 Thus, if a new path π_5 is to be added in step 3, it must be in the upper-left quadrant centered at π_4, namely, it must be shorter than π_4,

The key strategy of this algorithm is that we always retain for a pixel a list of paths with the NDP. A path to a certain pixel, N, that is more costly and longer than existing paths stored in N 's PATH-LIST need not be retained; extension of such a path can not produce a path of better quality than existing paths.

We now show that procedure *Find-Path* correctly computes PATH-LISTs using induction on n, the number of elements in all PATH-LISTs. We prove that for an element being added to a PATH-LIST, there exists no path that is strictly shorter and strictly economical than that element being added. Clearly, when n=1, the claim is true.

Suppose an element [K_i, P_j, e, l] is being added to P_j's PATH-LIST in Step 3. If this operation is not correct (i.e., it cannot be entered to P_j's PATH-LIST), then there must exist another element [M,P, e', l'] such that $e' \leq e$ and $l' \leq l$. This element must be either currently in the queue or a descendant of an element currently in the queue. In either case, the value e cannot be the smallest element in the queue. However, it contradicts the choice made in step 2, that is, e is the smallest element in the queue.

SIMULATION RESULTS

In this section, we show some results of software simulation of the multiobjective search algorithm described in the previous section. Figure 3a contains a synthetic example terrain in which the dark areas represent plateaus, the lines indicate valleys, and the black dot is the lowest point which also serves as the destination point. The start point is selected about where the three valleys meet in the center of the map. Figure 3b contains the result of running the multiobjective search algorithm in which the dark lines indicate paths. In this run, we measure distance by the Manhattan distance, while the cost of moving from pixel P to pixel Q is assigned as

$$cost(P,Q) = \begin{cases} \alpha(height(P) - height(Q)) & \text{if } height(P) < height(Q) \\ \beta(height(Q) - height(P)) & \text{otherwise} \end{cases}$$

where α and β are coefficients.

In this simulation, we set $\alpha = 3\beta$, indicating that climbing up consumes three times more energy than descent. There is no pixel that is not traversable in the scene. The result shows that there are two classes of paths, one corresponding to shortest paths (moving straight from the start point to the destination, traversing the plateau) and the other corresponding to paths with lowest economy (moving around the plateau). There is no third type of paths, meaning that there is no gain in traversing the plateau somewhere between the two path classes in this terrain. The results of the algorithm applied to scenes in Figs. 4a and 5a are shown in Figs. 4b and 5b, respectively. In these cases, there are three classes of paths.

The resolution of the pixel maps is 40 by 40 and planning the paths takes about 5 seconds on a Silicon Graphics Indigo. Most pixels have been explored by the search algorithm. Thus, running time of the algorithm is sensitive to the resolution of the map as well as to the number of objectives. For example, the algorithm takes on the order of several minutes to run for a map which is a couple of times longer than those illustrated.

DISCUSSION

Comparison

We will now compare our approach with other approaches that are often used in navigation. A popular approach for navigation in an obstacle-cluttered environment is the potential field approach [7, 8]. In this approach, the path is determined by repulsive forces received from the obstacles and attractive force from the destination point. The path tends to pass through the valley of potential field, thus the path produced is in general considered as safe. However, the path may lead into a local minimum depending on the shapes of the obstacles, at which point the vehicle on the path does not make any

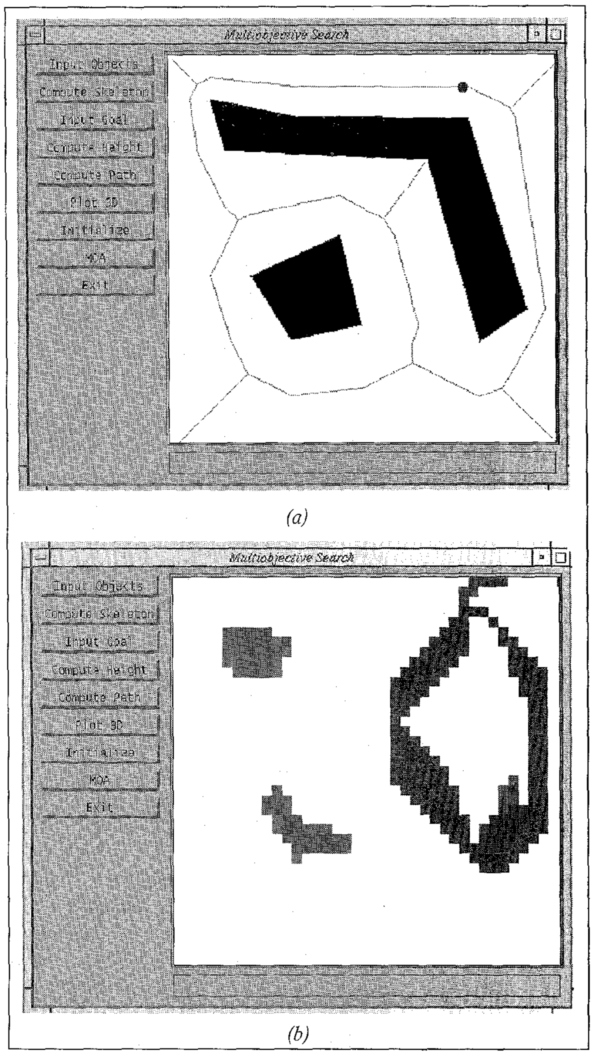


Figure 3.

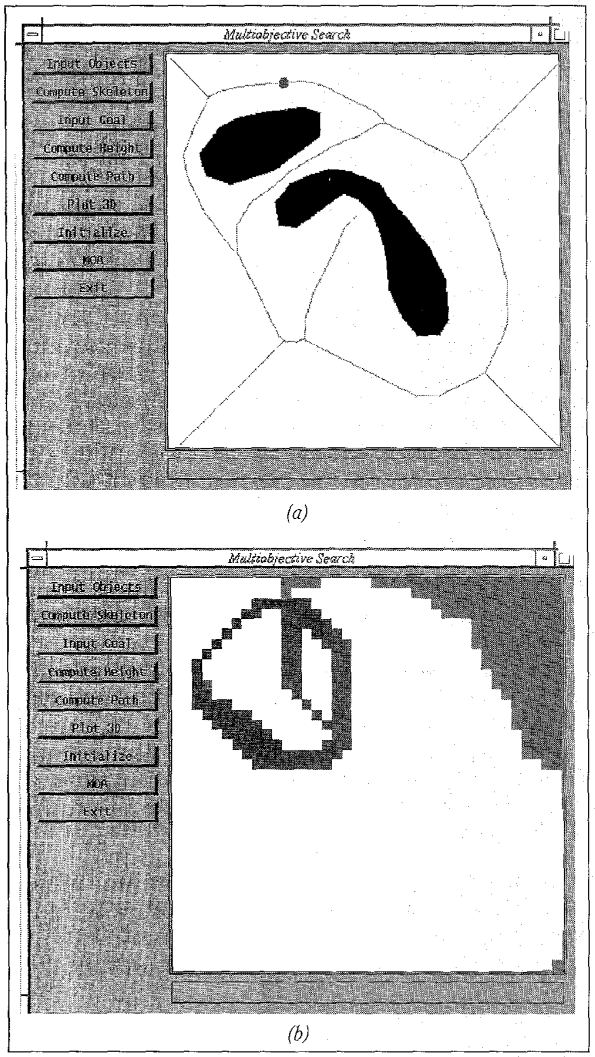


Figure 4.

progress toward the goal point. Methods using the Voronoi diagram [14] and skeletons [1] also keep the vehicle away from the obstacles.

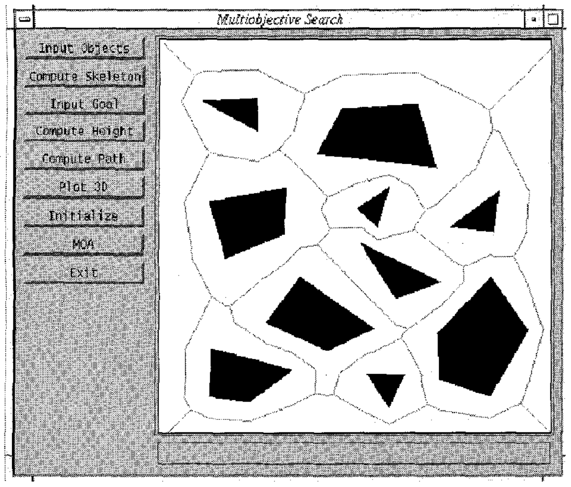
On the other hand, a path generated by the visibility graph [9] is known to be shortest and consists of vertex-to-vertex motions. Most approaches in terrain navigation based on grid representations [11, 2] use heuristic search methods such as A* to find a path. In this case, the path can be optimized in terms of some single criterion set by the user. Often, total path length is used to optimize the path. A method based on variational calculus [13] can take into account a blend of safety and path length.

The primary difference between our method and the approaches mentioned above is that our method generates a group of paths as opposed to a single path. This enables the user to select an appropriate path that suits his needs. On the other hand, it takes more time to execute the search procedure since we must keep track of all the nondominated paths at all times during search. For a single objective search, only one path needs to be retained at any pixel during the search, while in multiobjective search multiple paths need to be retained and processed.

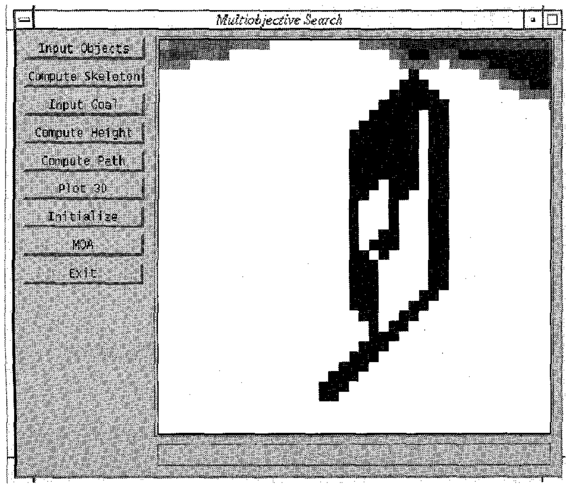
Extensions

So far we have assumed that the environment is static, that is, it does not change over time. In this section, we consider an extension of our approach to a time-varying environment. We consider the case in which the environment contains some obstacles whose locations change over time along known trajectories. The problem of path planning among moving obstacles has been approached by a variety of methods [3, 4, 6, 10].

At the beginning of this article, we stated that shortest paths and fastest paths are usually different in a time-varying environment. We note that there is a dependency between fastest paths and other paths in a time-varying environment. To see this, consider the case in which a person inside a certain room



(a)



(b)

Figure 5.

is to reach a point outside of the room through a door.

Suppose that he can move out of the room only when he makes the fastest move to the door. In such a case, in order to plan any path to reach the goal point, he first needs to make a fastest move to the door. Thus, we see that planning a minimum-delay path is a prerequisite for the problem of finding any path (optimizing any criterion) at all. Multiobjective search can be effectively used for such a case.

Now we extend our bitmap structure to handle moving objects. In a time-dependent environment, a pixel takes one of the two time-dependent states, "free" (not covered by any of the obstacles) or "occupied" (covered by some obstacle, thus not traversable). A pixel in the environment alternates between "free" and "occupied" states. Thus, we can keep a sequence of free time intervals for each pixel. The representation scheme for time may be regarded as run-length coding. The formulation can be used to represent arbitrary time-varying objects within a given resolution, rather than being able to represent just moving obstacles. It can represent obstacles whose boundary shapes change over time, etc.

Our path planning problem can be interpreted as finding a sequence of free intervals from a given start position to a given destination position under certain conditions concerning a speed constraint of the robot. The speed bound affects the travel time and choice of paths in the environment. Fastest paths and shortest paths can be computed by using a multiobjective search method analogous to the one described in the previous section. As noted above, in this case, we cannot use shortest path algorithms alone to find a path of minimum-length, as the shortest path problem contains minimum-delay path problem as a part of it. See [4] for details. After finding a path, we can postprocess the path by using, say, the method of [3] to produce a path satisfying its physical constraints such as acceleration limits.

CONCLUSIONS

We have considered a path planning approach that is suitable when the user has multiple criteria for path selection and wishes to choose an acceptable path from a set of candidate paths. We have used the concept of nondominant property in our path search and shown its feasibility in case of two objectives by computer simulation. A navigation path is typically acceptable when it satisfies multiple objectives. The presented search algorithm is capable of satisfying this requirement. As a natural consequence, its processing time is multiple times slower than search algorithms for a single objective. The queue size will limit the range of applications. A method using a hierarchical search [5] may have to be used to speed up the approach.

ACKNOWLEDGMENTS

The author would like to thank the participants of seminar cis788.01I at Ohio State University for their helpful discussions.

REFERENCES

[1] J. Barraquand and J. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research,* 10:72 89, 1990.

[2] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computers,* 22:46 57, 1989.

[3] T. Fraichard and C. Laugier. Path-velocity decomposition revisited and applied to dynamic trajectory planning. In *Proceedings of the IEEE International Conference on Robotics and Automation,* pages 40-45, 1993.

[4] K. Fujimura. Motion planning using transient pixel representations. In *IEEE International Conference on Robotics and Automation,* pages 34-39, 1993.

[5] K. Fujimura and H. Samet. A hierarchical strategy for path planning among moving obstacles. *Proceedings of the IEEE Transactions on Robotics and Automation,* 5:61-69, 1989.

[6] N.C. Griswold and J. Eem. Control for mobile robots in the presence of moving objects. *Proceedings of the IEEE Transactions on Robotics and Automation,* 6:263-268, 1990.

[7] Y.K. Hwang and N. Ahuja. Potential field approach to obstacle avoidance. *Proceedings of the IEEE Transactions on Robotics and Automation,* 8:23-32, February 1992.

[8] J. Kim and P.K. Khosla. Real-time obstacle avoidance using harmon-

ic potential functions. *IEEE Transactions on Robotics and Automation,* 8(3), June 1992.

[9] T. Lozano-Perez and M.A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM,* 22:560 570, 1979.

[10] C. Shih, T. Lee, and W.A. Gruver. A unified approach for robot motion planning with moving polygonal obstacles. *IEEE Transactions on Systems, Man, and Cybernetics,* 20:903 915, 1990.

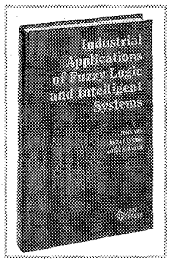
[11] A. Stenz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation.pp.* 3310-3317, 1993.

[12] B.S. Stewart and C.C. White III. Multiobjective A*. *Journal of the ACM,* 38(4):775-814, 1991.

[13] S. Suh and K. Shin. A variational dynamic programming approach to robot path planning. *IEEE Transactions on Robotics and Automation,* 26:334-349, 1988.

[14] O Takahashi and R.J Shilling. Motion planning in a plane using generalized Voronoi diagrams. *IEEE Transactions on Robotics and Automation,* 5: 143-150, 1989.

Kikuo Fujimura received the B.S. and M.S. degrees in information science from the University of Tokyo, Japan, in 1983 and 1985, respectively, and the Ph.D. degree in computer science from the University of Maryland, College Park in 1990. He was with the Autonomous Robotics group of the Oak Ridge National Laboratory from 1990 to 1991, where he worked on mobile robot navigation. In 1991, he joined the Department of Computer and Information Science of the Ohio State University, where he is currently an Assistant Professor. His research interests include computer graphics, vision, and machine intelligence.



Learn how to effectively incorporate fuzzy logic control in the design of intelligent systems

INDUSTRIAL APPLICATIONS OF FUZZY CONTROL AND INTELLIGENT SYSTEMS

edited by Dr. John Yen, Dr. Reza Langari, Texas A&M University, and Dr. Lotfi A. Zadeh, University of California at Berkeley

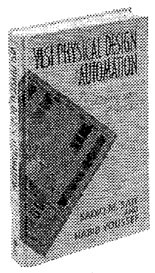
This timely and extensive text focuses on industrial applications of fuzzy logic with particular emphasis on real functioning systems. A comprehensive volume of particular interest to industrial practitioners who want to explore real world fuzzy and intelligent systems applications.

April 1995/Hardcover/376pp
List Price: $64.95 · **Member Price: $52.00**
IEEE Order No. PC4002-QBZ · ISBN 0-7803-1048-9

ORDER 24 HOURS A DAY • 7 DAYS A WEEK!

Call 1-800-678-IEEE (toll-free, USA and Canada)
or **1-908-981-0060** or **Fax 1-908-981-9667** or mail to:

IEEE The Institute of Electrical and Electronics Engineers, Inc.
445 Hoes Lane, PO Box 1331, Piscataway, NJ 08855-1331 USA



A comprehensive introduction to VLSI — designed expressly for progressive learning...

VLSI PHYSICAL DESIGN AUTOMATION: THEORY AND PRACTICE

by Sadiq M. Sait and Habib Youssef, King Fahd University of Petroleum and Minerals - Saudi Arabia

co-published with McGraw-Hill Europe

Newly recognized as an increasingly important area of electrical and computer engineering, VLSI has, until now, been unavailable in such an accessible textbook form. The authors approach each topic with a solutions-driven format ideal for senior undergraduates, graduates, and anyone starting work in the field of CAD for VLSI. Covering all aspects of physical design, this book is complete with illustrated solutions and examples formatted by: problem definition, cost functions and restraints, possible approaches, and latest developments.

April 1995/Hardcover/448pp
List Price: $54.95 · **Member Price: $44.00**
IEEE Order No. PC5609-QBZ · ISBN 0-7803-1141-8

ORDER 24 HOURS A DAY • 7 DAYS A WEEK!

Call 1-800-678-IEEE (toll-free, USA and Canada)
or **1-908-981-0060** or **Fax 1-908-981-9667** or mail to:

IEEE The Institute of Electrical and Electronics Engineers, Inc.
445 Hoes Lane, PO Box 1331, Piscataway, NJ 08855-1331 USA