Proceedings of the 2003 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
Las Vegas, Nevada - October 2003

Flexible Binary Space Partitioning for Robotic Rescue

Jacky Baltes and **John Anderson**
Autonomous Agents Laboratory
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba
Canada R3T2N2
Email: jacky,andersj@cs.umanitoba.ca

Abstract

In domains such as robotic rescue, robots must plan paths through environments that are complex and dynamic, and in which robots have only incomplete knowledge. This will normally require both diversions from planned paths as well as significant re-planning as events in the domain unfold and new information is acquired. In terms of a representation for path planning, these requirements place significant demands on efficiency and flexibility. This paper describes a method for flexible binary space partitioning designed to serve as a basis for path planning in uncertain dynamic domains such as robotic rescue. This approach is used in the 2003 version of the Keystone Fire Brigade a robotic rescue team. We describe the algorithm used, make comparisons to related approaches to path planning, and provide an empirical evaluation of an implementation of this approach.

Introduction: Path Planning in Robotic Rescue

Navigation is an important task for any mobile robot, since being at the right place at the right time is a precursor to performing almost any useful task. Navigation in very simple domains can be done purely reactively (for example, the reactive foraging of Balch (Balch & Arkin, 1994)). In such domains, a robot can be successful despite minimal navigation abilities simply because being in the right place at the right time is not particularly important: If we are foraging for targets, we will eventually find them all if we simply wander enough. If, however, we are trying to find all of them in a minimal amount of time, it helps to keep track of where we have been, where we are, and where we are going. The majority of real-world tasks are similar: in a domain that is complex or hazardous, planning ahead to some degree both increases the likelihood of success and the likelihood that the robot will survive its travels.

Navigation in a domain of any sophistication requires a number of important sub-tasks: localization, path planning, and plan execution. Localization involves maintaining some concept of one's position in the overall environment, and goes hand in hand with the other sub-tasks. Path Planning is the problem of creating a collision-free path through a set of obstacles from an initial to a goal position. Finally, plans must be carried out.

Robotic rescue environments represent one of the most difficult challenges a path planner can face. To measure progress in the field of robotic rescue, international competitions are held using a number of reconfigurable environments following a set of standards (e.g. (Jacoff, Messina, & Evans, 2001)). In this task domain, it is certainly possible to have a purely reactive robot searching for victims with no knowledge of the area. We have previously implemented such an approach (Baltes & Anderson, 2002). This experience has taught us that at absolute minimum, rudimentary path planning abilities are necessary. Being able to plan even a general path using limited knowledge can assist the robot in tasks such as finding the entrances to rooms, ways around obstacles, etc., that are beyond the capabilities of a purely reactive robot. During the course of rescue activities, building a map to victims is a primary concern. A robot should be able to take advantage of such a structure as a basis for further navigation, which also requires path-planning skills.

In the robotic rescue domain, knowledge about the environment will be both uncertain and dynamic. As such, it cannot be relied upon as the sole basis for navigation. An agent must be able to plan tentative paths and have real-time means of following these paths and dealing with errors in sensing and odometry that are necessary evils in the real world. It must also be able to detect when differences between a planned path and the real world are significant enough to be beyond minor adjustments, and decide when to re-plan based on this new knowledge. The fact that re-planning must be done often and in real time places very hard demands on a path planning approach in terms of computational efficiency. The fact that we may wish to try to salvage parts of a plan also places demands on a path planner in terms of flexibility of representation: a more flexible representation of the space in which path planning was performed will allow greater exploration of alternatives based on a currently non-executable plan given real time constraints.

The Keystone Fire Brigade is a robotic rescue team

0-7803-7860-1/03/$17.00 © 2003 IEEE

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 10, 2008 at 14:52 from IEEE Xplore. Restrictions apply.

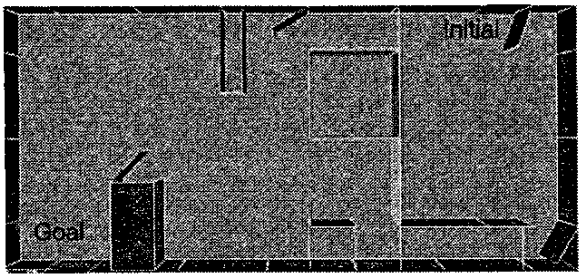

Figure 1: A Sample Path Planning Problem in the RoboCup Rescue domain

that has competed in the 2002 competitions in both Fukuoka (RoboCup) and Edmonton (AAAI). The key elements of our approach are emphasis on autonomy, vision-based computing, and implementation on inexpensive robot bases (toy car platforms). The 2002 Keystone Fire Brigade team employed a simple reactive navigation and heuristic mapping implementation. We have since been exploring the use of flexible variants of binary space partitioning as alternatives for path planning in such domains. In this paper, we describe an improved approach to real-time path planning based on these ideas for use in domains such as robotic rescue. The remaining sections of this paper reviews traditional approaches to path planning, describes our approach to flexible binary space partitioning, describes the implementation of this approach for the 2003 Keystone Fire Brigade team, and provides an empirical evaluation.

Previous Approaches to Path Planning

The path planning problem is formally defined as: given an input location, a goal location, and a description of a set of obstacles, create a collision free path that will move the robot from the initial position to the goal position. A sample path planning problem taken from the NIST testbed is shown in Figure 1. The figure does not include the dynamic objects, nor the extensive debris that is typically placed throughout the testbed. While this might be typical for plotting a path through a room to the next, paths based on desired area coverage are also important - in that case, the goal would be in the general vicinity of an unexplored region of the environment rather than a specific point such as a door.

There have been many different approaches to path planning for both holonomic and non-holonomic robots. The major approaches for holonomic path planners can be divided into skeletonization methods (such as visibility graphs, Voroni diagrams, or road maps), cell decomposition (e.g., approximate or exact quad-tree decomposition), or local approaches (e.g., potential fields, landmarks) (Russell & Norvig, 1995; Latombe, 1991).

Skeletonization Methods

Visibility graphs are the most commonly used skeletonization method. In path planning using visibility graphs, the planner constructs a graph from its representation of the environment. In this graph, the edges of the graph are the edges of obstacles, with the vertices being the intersections of edges, along with the start and goal points for the robot. Once this graph is constructed, the robot can then easily plot a path by adding vertices from the start and end points to those of nearby obstacles, and connecting disjoint obstacles in the graph through additional edges such that the cost of traversing each of those edges is minimal. It can be shown that the minimum length path from start to goal lies along the visibility graph (Latombe, 1991).

While theoretically very useful, visibility graphs are more problematic in practice. In a situation where there are obstacles to be traversed, the path produced by this approach follows the edges of the walls and obstacles that make up the visibility graph. Even if it can be assumed that the initial locations of these obstacles were extremely accurate, this results in a strong likelihood of the robot running into and against walls and other obstacles, due to small errors in motor encoding and localization.

Local Approaches

One alternative to working with skeletonization methods is to construct a representation that allows decision-making at run time to proceed using entirely local information. Potential field methods are representative of this alternative.

A potential field map defines an attraction (or repulsion) gradient across a spatial area using an arbitrary potential function(Arkin, 1998). A potential field map can be constructed for any environment in which an appropriate potential function can be defined, and paths are followed easily by moving to across the gradient of greatest potential. Decision making is entirely local: a robot in any particular location calls the potential function to take into account the perceived or known forces on it at that point in time, thereby defining the gradient value for that areas around the particular point the robot occupies. This method involves weaker assumptions of accuracy compared to visibility graphs, and is more dynamic in nature. However, because the agent is simply following a gradient toward desirable states and away from undesirable states, this method suffers from the same problem of all gradient-descent problems: local minima/maxima. To deal with these, changes must be made to the potential field at run time for the robot to be able to escape any local minimum/maximum states. Detecting the locations of these states and repairing them, however, is not a trivial problem.

Cell Decomposition Methods

The third common approach is that of cell decomposition, such as quad-tree or oct-tree decomposition or

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 10, 2008 at 14:52 from IEEE Xplore. Restrictions apply.

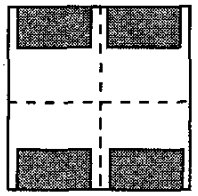

Figure 2: Space Partitioning in a Quad-Tree Approach

binary space partitioning (e.g. (Zelinsky, 1992; Saona-Vazquez, Navazo, & Brunet, 1999)). Cell decomposition methods build a representation of the environment for path planning by recursively decomposing the known or detectable spatial area into partitions of regular size. Each approach begins with a single large cell, and labels that cell as either blocked (i.e. entirely filled by an obstacle), free (completely open space), or mixed (at least part of the cell is occupied by an obstacle). Any cell that is mixed must be recursively broken down by sub-partitioning it, resulting in a tree structure. Any cell that can be labelled as blocked or free at any point forms a leaf node in that tree structure. As this tree is built, adjacency links must be maintained between branches of the tree in order to facilitate path planning.

The major variation between cell decomposition mechanisms is the number of recursive partitions that are generated from any area labelled as mixed: a quad-tree decomposition breaks the space into four regular partitions, while an oct-tree decomposition breaks it into eight, and a binary space partitioning process into two. These approaches thus trade off the depth of the resulting tree with the number of potential adjacency links that must be maintained and the work involved in partitioning the domain. In addition to this trade-off, each approach relies on a precise, regular means of partitioning a space.

Once the entire space is broken down into blocked and free cells, path planning can ensue by exploring open spaces using adjacency links. Algorithms for producing plans are reasonably simple and efficient once the environment is decomposed. The plans produced are practical to execute at runtime, and there is no issue of local minima at runtime. However, there are issues of complexity in quad-tree decomposition that currently limit its applicability to path planning. In performing decomposition it is non-trivial to manage the tree structure and appropriately insert adjacency links between regions that may be in physically very distant parts of the tree.

More importantly, the issue of partitioning space evenly makes decomposition more work than is necessary and results in trees that are less efficient representationally that is desirable. Consider the situation in Figure 2. The overall area has approximately 50% obstacle coverage. However, using a quad-tree decomposition, all areas are still labelled as mixed after one decomposition, forcing an additional level in the tree and a corresponding increase in nodes and adjacency links. A common-sense decomposition would identify the large open areas in the middle of the path. In any cell-decomposition approach we must choose an atomic level of interest, and for path planning this should be approximately the largest dimension of the robot itself. However, situations like the one above force a decomposition to use a significantly smaller granularity (e.g., half the robot width), with corresponding increase in complexity and resource consumption.

Because it is recursive in nature, quadtree decomposition can be viewed as an anytime approach from the point of view of representation: we begin with a very coarse breakdown and gradually make it finer given more time to compute. From the point of view of path planning, however, this does not induce the ability to plan in an anytime manner: a path plan cannot be constructed until the decomposition is complete.

It is these overhead issues that make the use of visibility graphs the most popular choice in contemporary path planners, despite the fact that they produce inferior plans from an execution standpoint. However, this overhead could be reduced if the cell size that was necessary to decompose to were made larger, and more importantly if it were possible to identify the larger open areas to begin with (and thus not have to break down portions of the tree so deeply).

The next section describes *Flexible Binary Space Partitioning*, a method based on cell-decomposition that allows us to avoid using the poorer paths induced by visibility graphs, while limiting the overhead associated with cell-decomposition approaches.

Flexible Binary Space Partitioning

The major problem with the use of cell decomposition methods in path planning is that in general they forgo the overhead of making intelligent decisions in partitioning in order to make the decomposition process more efficient. It is our view that this is a poor trade-off, and that an intelligent partitioning mechanism can be applied that will more than make up for its overhead through the development of trees that are smaller and more effective for path planning.

What is required, ultimately, is a heuristic for partitioning that is computationally feasible to calculate yet still biases the partitioning process to identifying open spaces quickly and allow those to be grouped together rather than appearing in isolated regions of the tree. While the characteristics of such a heuristic are similar irrespective of the cell decomposition method employed, our approach is based on binary-space partitioning. This is because quad-tree, oct-tree, and larger decompositions are sub-optimal when the decomposition method does not result in square regions of roughly equal size. This will occur, for example, when partitioning a long, thin room. By attempting to select partitioning points that will identify open and blocked spaces quickly, we largely guarantee there will be no

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 10, 2008 at 14:52 from IEEE Xplore. Restrictions apply.

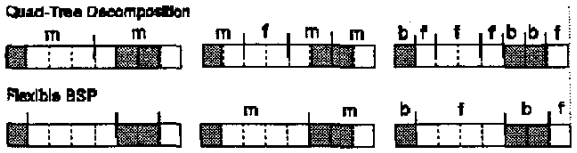

Figure 3: Decomposition of a one-dimensional example space

regular square regions. At the same time, any quad-, oct-, or larger decomposition can be represented by a series of binary space partitions, meaning an approach based on BSP will subsume these others.

We motivate our approach using a simple one-dimensional problem illustrated in Figure 3. The top portion of the figure illustrates a quad-tree decomposition of the example space. While a quad-tree decomposition breaks each cell into four equal-sized partitions, this view shows only one dimension of the space, and so two partitions are made. Both of these partitions are mixed, and so a second iteration splits these as well. The second iteration identifies one completely free partition, and the others all must be split by a third iteration before all partitions are labelled as free or blocked. In a two-dimensional space, this would simply occur over both dimensions. Note that the space partitioning causes two problems: one large four-cell gap is split into three separate pieces in different branches of the tree, and the right obstacle is split over two cells as well. For a robot more than two cells wide the former problem would cause difficulty during path planning, since space must be amalgamated before a path could be plotted through this region. Similarly, the large obstacle is essentially viewed as two smaller obstacles, which may also cause problems when trying to predict the motion of obstacles.

The difficulty here is that the openings and obstacles in the domain are not recognized for what they are. An appropriate heuristic must begin by attempting to separate these areas in a useful way rather than splitting at a fixed point. The lower portion of Figure 3 begins with the identification of the points marking the transition between blocked and open spaces. These mark three *potential* places where a partition could be made. The heuristic we employ requires only that we test these three points, rather than all of the points in the space; this is explained in detail below.

Having a fixed split point does not guarantee the introduction of any new knowledge regarding the resulting partitions: while we will certainly eventually break the space down eventually, the process of performing one subdivision does guide the process toward the identification of useful spaces. The main motivation for our algorithm is that any split in the space should result in a high likelihood of *more* knowledge about the two resulting partitioned regions - the times where there are no new partitions that can be labelled should be exceptions rather than the rule. This can be discussed in terms of *entropy* - the minimum length of a code describing the area, which can also be viewed as the percentage of mixture of the resulting regions. We calculate a heuristic value to associate with each potential split point by comparing the size and entropy of the regions that would result if that point was used to partition the space (i.e., weighted entropy). In the example in Figure 3, the point that results in the least entropy is the middle of three potential partition points. Both of the new partitions that result are labelled mixed, and like other cell decomposition methods, this now continues recursively. In the next stage, each of the left and right partitions has only one choice for a potential partition point, and so these are used. The resulting set of partitions is much better than those produced by the quad-tree decomposition: a large open block of four has been recognized, and no obstacles are divided between partitions. The resulting decomposition tree is also smaller and easier to employ in path-planning.

There are two parts to solving this problem: identifying the potential partition points, and selecting the partition point that would result in the most useful partitions were it to be used in cell decomposition. We will deal with the latter first.

When describing information content of a signal, the concept of entropy - originally introduced by Shannon (Shannon & Weaver, 1949) - is often used. Entropy refers to the minimum description length required to encode a message, but has also been applied in other areas requiring the measuring of information. For example, the ID3 heuristic for learning a decision tree uses the concept of information gained by splitting the decision tree on a particular attribute (Mitchell, 1997). Similar measurements based on entropy have been applied in other areas as well, such as Balch's (Balch, 1998) social entropy, measuring the information imparted to a group by a heterogeneous member.

We define the entropy of a spatial area as

$$\text{Entropy}(A) = -p_f \log_2 p_f - p_b \log_2 p_b$$

Where

$$p_f = \frac{|f|}{|A|}$$

and

$$p_b = \frac{|b|}{|A|}$$

are the percentage of free and blocked areas respectively. The entropy function measures the information in a given space through the relative proportion of free and blocked areas. This means that the more evenly mixed the free and blocked areas in the space are, the more entropy is present and less information is provided by that space. Note that since entropy is a probabilistic measure of the complexity of a domain, it is particularly suited to highly-dynamic uncertain domains such as robotic rescue. It is also inherently robust to sensor noise.

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 10, 2008 at 14:52 from IEEE Xplore. Restrictions apply.

In order to estimate the gain in information we might achieve by partitioning an existing space using a potential partition point, we must look at the difference between the entropy of the original space and the total entropy in the resulting partitioned spaces, with each of the latter weighted by its proportional size:

$$\text{gain}(p_i, A) = \text{E}(A) - \sum \frac{|P|}{|A|}\text{E}(|P|)$$

Information gain is the factor used to choose the best potential partitioning point. It would appear that the most computationally challenging part of the problem would be the calculation of potential entropy for each and every cell that could potentially be used as a partitioning point. However, as can be seen from the formulae above, the gain in information depends strongly on the percentages of blocked and free areas. This means that *significant* changes in entropy will only occur when we split at the junction of a free and a blocked cell. So, in a given space, we need only perform entropy calculations on such transition points - for example, the three potential partition points identified in lower left portion of Figure 3.

While the motivating example was one-dimensional for simplicity of explanation, in the more common two-dimensional cases (or higher dimensions) the same method may be applied. In a two-dimensional case, the gain is calculated for each partitioning point as well as partitioning direction. That is, each point is considered along either dimension.

The other manner in which the motivating example has been simplified is in the nature of the free and blocked regions: in simple a one-dimensional vector, the obstacles are necessarily rectangular. In a real-world situation, however, blocked regions will be irregular and can be approximated by polygons. Figure 4, for example, contains two polygons that are not aligned to the coordinate space. In this case, the vertices of the polygons are projected onto the dimension of partitioning, as shown in the figure. While the worst-case projection is only linear in the number of vertices, this is still overhead that a simple regular partitioning method would not require. However, in cases where there is a large number of vertices, this complexity can be reduced by selecting only a subset of these vertices (which is also being done by approximating the irregular obstacles with polygons to begin with).

Empirical Evaluation

An implementation of our method was compared against the standard quad-tree decomposition path planning algorithm using C++ under Linux. A sample run of this implementation in comparison with a standard quad-tree decomposition is illustrated in Figure 5. In this problem, 16 obstacles of the same size were arranged in a regular pattern. The decomposition of this space resulted in a tree of 311 leaf nodes for a quad-tree decomposition, vs. 122 for using the flexible

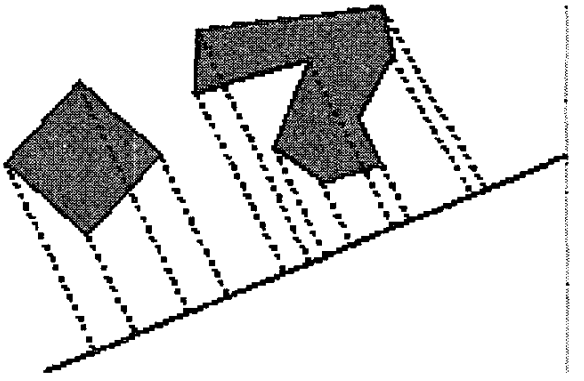

Figure 4: Representing polygons by projecting their vertices

BSP approach detailed in the previous section. Beyond having a simpler tree to use in path planning, one can see that there are in general more useful open spaces and the spatial decomposition is much more intuitive. This is particularly true for the right-hand side of the environment, where there is more open space and the representation is less disjoint. These results are representative of the quality of the decomposition - for simpler environments we obtained even better results (e.g. an 11 vs. 41 node tree for a two-obstacle environment). To examine this over a range of problem complexities, we generated random test problems over a 2.8m x 2.3m test environment. The size as well as the position of the obstacles was created at random, and the quality of the decomposition in terms of the number of leaf nodes in the resulting tree was examined. The number of obstacles in test problems ranged from 1 - 32. The results of this, over increasing obstacle numbers, is shown in Figure 6. These experiments were re-run several times, and in each case the results were consistent. Considering the decompositions produced for all problems, the flexible BSP approach generated 27% of the nodes generated by a quad-tree decomposition.

Conclusions

It is non-trivial to develop a good path planning method for a domain as complex as robotic rescue. The results obtained by our experiments suggest that flexible BSP is a very promising alternative for decomposing a representation of a region for the purposes of path planning. We believe this approach to be especially feasible for use in the robotic rescue domain (and similar highly dynamic and uncertain domains) because it is based on probability estimates rather than relying on a complete precise model of the domain. While re-planning of paths will always be necessary in dynamic domains, the flexible BSP approach produces simpler trees that make path planning itself a simpler task, which in turn facilitates more effective re-planning when necessary (and

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 10, 2008 at 14:52 from IEEE Xplore. Restrictions apply.

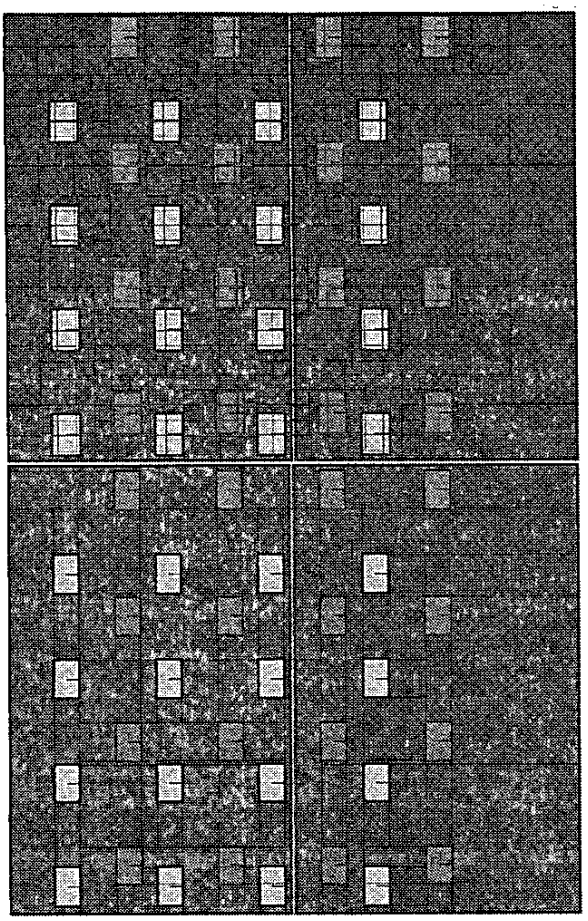

Figure 5: Quad-tree decomposition vs. flexible BSP

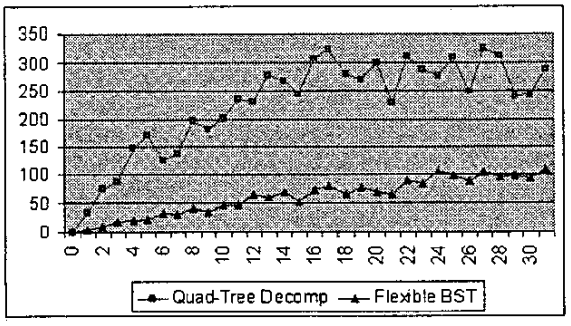

Figure 6: Comparison of approaches over increasing number of obstacles

allows more re-planning given the same computational resources).

We will be employing this approach in the 2003 Keystone Fire Brigade our entry for the upcoming year's robotic rescue competitions. There two elements central to the design of the Keystone Fire Brigade that we believe make this approach particularly suitable. First, the Keystone Fire Brigade is based on inexpensive toy car platforms, allowing only extremely limited odometry. We expect that this mechanism will allow more extensive path planning while still not requiring a high degree of odometry accuracy, because of the simpler decomposition. In additional to limited odometry, the Keystone Fire Brigade possesses only limited computational abilities, much of which is already committed to visual information processing. The limited processor speed and environment require a method that will provide smaller, more efficient data structures for path planning. We believe flexible BSP will be an ideal approach under these constraints.

References

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

Balch, T., and Arkin, R. C. 1994. Communication in reactive multiagent robotic systems. *Autonomous Robots* 1(1):27–52.

Balch, T. 1998. *Behavioural Diversity in Learning Robot Teams.* Ph.d., Georgia Institute of Technology.

Baltes, J., and Anderson, J. 2002. A pragmatic approach to robot rescue: The keystone fire brigade. In *Proceedings of the Robotic Rescue Workshop, International Joint Conference on Artificial Intelligence (IJCAI-02).*

Jacoff, A.; Messina, E.; and Evans, J. 2001. Experiences in deploying test arenas for autonomous mobile robots. In Messina, E., and Meystel, A., eds., *Proceedings of the 2nd Performance Measures for Intelligent Systems Workshop (PerMIS)*, NIST Special Publication 982, 87–95. National Institute of Standards and Technology.

Latombe, J.-C. 1991. *Robot Motion Planning.* Boston: Kluwer Academic Publishers.

Mitchell, T. 1997. *Machine Learning.* McGraw-Hill.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, New Jersey 07632: Prentice-Hall Inc. chapter 20, 598–624.

Saona-Vazquez, C.; Navazo, I.; and Brunet, P. 1999. The visibility octree: A data structure for 3d navigation. *Computer & Graphics* 23(8):635–644.

Shannon, C., and Weaver, W. 1949. *The Mathematical Theory of Communication.* Urbana: University of Illinois Press.

Zelinsky, A. 1992. A mobile robot navigation exploration algorithm. *IEEE Transactions of Robotics and Automation* 8(6):707–717.

Authorized licensed use limited to: IEEE Xplore. Downloaded on November 10, 2008 at 14:52 from IEEE Xplore. Restrictions apply.