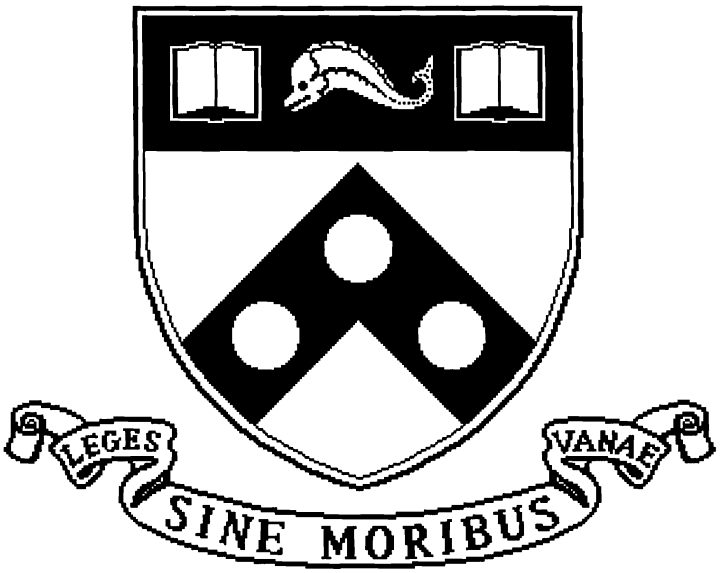
Exploratory Aspects of Sensor Based Planning

MS-CIS-98-07

Andrew Hicks, David Pettey



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

1998

Exploratory Aspects of Sensor Based Planning

Technical Report MS-CIS-98-07

Andrew Hicks

GRASP Laboratory
Department of Computer
and Information Science
University of Pennsylvania
Philadelphia, PA 19104
rah@grip.cis.upenn.edu

David Pettey

Department of
Physics and Astronomy
University of Pennsylvania
Philadelphia, PA 19104
pettey@student.physics.upenn.edu

Abstract

In sensor based planning exploration is unavoidable. To understand this aspect of sensor based planning, the authors consider the problem of motion planning for a point with "tactile sensors". In dimensions greater than two, this problem has been shown to be unsolvable given a certain mathematical framework. But, if the formulation of the problem is changed by taking the C-space to be discrete, then path planning with tactile sensors is possible. In this setting we give a resolution complete algorithm for planning the motion of a point in any dimension. Measuring the complexity of the problem by the number of discrete moves that the robot makes, we give an upper bound for the complexity of our algorithm that is linear in the surface area of the boundary of the C-space obstacles.

1 Introduction

Classical robot motion planning considers problems where the robot has full knowledge of it's workspace. Unfortunately, robots are sometimes faced with the more difficult situation of having incomplete descriptions of their environment, and consequently the classical algorithms are often not applicable. Often, robots have only *local* knowledge of their environment, i.e. their sensors have limited range and the problem becomes one of *sensor based motion planning.* A familiar example of this is when a robot's only sensor is a camera, and so only line-of-sight information is available. A more extreme example is *tactile sensing*, in which case the robots sensors can only detect an object by having physical contact with it[1]. Using tactile sensors, the robot gains information about the environment by "groping" around the environment, like a person searching for an object in a dark room. The major difference between classical planning problems and sensor based planning problems is that sensor based planning problems invariably have an exploratory aspect to them, even if the robot has the exact coordinates of it's goal (see figure 1). The amount of exploration necessary depends, of course, on the type of sensor: one would expect a robot with only tactile sensors to take much longer to reach it's goal than a robot with a camera.

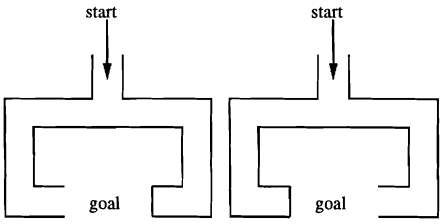


Figure 1: A robot with vision must get to the goal. Even if it knows the coordinates of the goal, it does not know which of the two above situations it is in. Thus forced exploration is an inevitable problem in sensor based planning.

The authors goal was to understand the ex-

[1]Throughout this paper we will use the term "tactile" to include very short-range sensors, i.e. the robot may not have to actually touch an object to sense when it is very close to it.

ploratory aspects of sensor based planning. We chose to investigate planning with tactile sensors because we feel that the problem of sensor based planning should be viewed as being made of a collection of local problems which are patched together according to global data. This data must be obtained from exploration. Thus, when investigating the exploratory nature of the problem, the fundamental unit that is sensed is not important. The more general problem of efficiently fusing together all of the local information obtained by more complex sensors will be addressed in future work.

2 Contributions

We give an algorithm, *Makepath*, for tactile planning in a discrete configuration space of any dimension. This algorithm amounts to an on-line search on a on the boundary of the C-space obstacles, similar to what is proposed in [10] and also similar in spirit to [5].

Makepath generates a path in less than $2S + d$ actual physical motions of the robot, where S is roughly the area of all of the C-space obstacles, measured in the fundamental unit of the resolution and d is the "Manhattan" distance from the start to the goal, also measured in the fundamental unit of the resolution. (This bound represents the "external complexity" of the problem, i.e. how much time must be spent by the robot moving.) It seems likely that $2S + d$ would be the best upper bound that could be achieved in our framework.

3 Relations to Previous Work

This paper has it's origins in two subjects: sensor based planning with tactile sensors and classical planning using discrete spaces. [2]

In [14], Lumelsky and Stepanov considered the problem of navigating a point in a plane with tactile sensors, and gave two algorithms to solve this problem. These algorithms produce paths whose lengths are no more than the sum of the straight line distance between the start and the goal, plus a term that was proportional to the total perimeter of the obstacles. Lumelsky and Sun, in [15] and [17], and Cheung and Lumelsky in [2], attempt to generalize this solution to higher dimensions. In [9], Kutulakos, Lumelsky and Dyer describe a theoretical framework for the problem of planning the motion of a point with sensors in n dimensions. They conclude that the problem of planning for a point with tactile sensors is necessarily unsolvable.

The discretization of the configuration space of a planning problem essentially reduces it to a graph search problem, as was observed by Donald in [5] and in [6]. Donald considers the problem of global path planning and develops the "Bumble Strategy", which was a breadth-first search from the start to the goal on a C-space grid. Here the C-space grid is internally available to the algorithm, which is what makes the problem global. In principle this algorithm could be implemented for on-line planning, but due to the fact that the robot would actually have to move to execute the search, the runtime would be too large because of the amount of backtracking done by the robot. This leads us to look for algorithms that minimize backtracking while searching the special types of graphs that arise from C-space obstacles.

4 The *Bug2* Algorithm

Makepath was motivated by Lumelsky and Stepanov's *Bug2* algorithm [14], which is for planning the motion of a point with tactile sensors in the plane, and has the very nice property of being provably complete. One major difference between our algorithm and *Bug2* is that our algorithm remembers the C-space obstacles that it encounters, in contrast to *Bug2*, which only uses a small amount of memory. *Bug2* can be described roughly as follows: travel on the straight line, l, that connects the start and the goal until bumping into an obstacle at a point p. Then follow the perimeter of the obstacle clockwise until encountering a point on l that is closer to the goal than p, which we will call an *exit point* (if a path connecting the start and goal exists, then this is guaranteed to occur because of the simple topological structure of planar obstacles). Then continue on the line towards the goal and repeat this process if necessary. See figure 2.

In the case of a point moving in n-dimensions, we can still move along a line l that connects the start and goal configurations in the configuration space. But now the obstacles encountered in the configura-

[2] Indirectly related, is the work that has been done with range data, such as [3], [8], and [7]. Additionally, there has been some work in the algorithms community done on the complexity issues of planning for a point in the plane, for example [16] and [1].

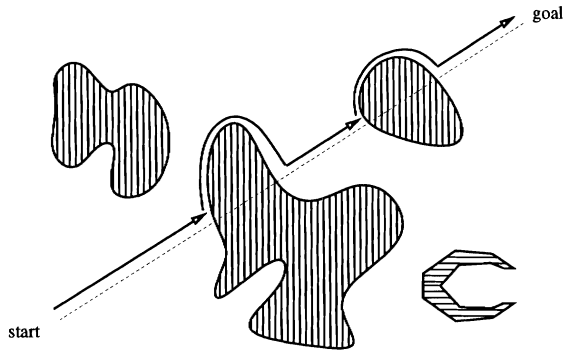


Figure 2: Lumelsky and Stepanov's Algorithm for navigating a point in the plane: head towards the object until bumping into it, then turn left, and follow the perimeter of the obstacle.

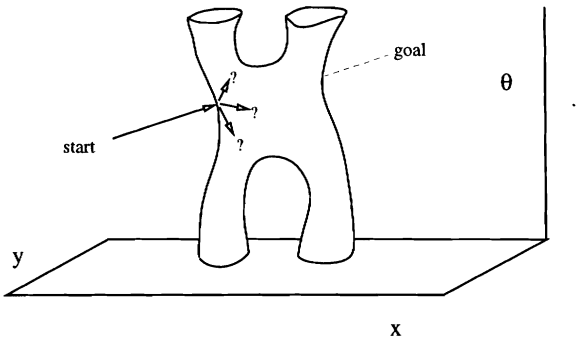


Figure 3: What do we do in higher dimensions ? The obstacles encountered in three dimensions are surfaces, and there is no natural way to go around them, as there was in the case where the robot is a point in the plane.

tion are manifolds of any dimension from 0 to $n-1$, and this is where the hard part of extending the algorithm begins. There is no notion of clockwise as there was in the lower dimensional case, and so there is no single direction to move in that will guarantee that the robot will find a closer point on l (see figure 3). Consequently, we are forced to find a way to search the surface, and we clearly we would prefer to do this with as little "backtracking" as possible in order to minimize the travel time for the robot.

5 Definitions

Since we will be making a discrete approximation to a continuous problem it is important to carefully define the objects that we will use.

Definition 1 A *(unit) cube* in R^n is a set of the form $[a_1, b_1] \times ... \times [a_n, b_n]$ where $b_i - a_i = 1, i = 1...n$. A *hyperface* of this cube is a set of the form $U_1 \times ... \times U_n$ where $U_i = \{a_i\}$ or $\{b_i\}$ or $[a_i, b_i]$. The number of intervals occurring in the product is the *dimension* of the hyperface. A hyperface of dimension $n-1$ is called a *face* and a hyperface of dimension $n-2$ is called an *edge*.

We take as our total space, T, the set of cubes of unit size whose vertices lie on points in R^n that have only integer entries. Thus each cube in T is a single state, and it does not make sense to consider moving around in the interior of a single cube. Two distinct cubes may intersect in a face, but they still correspond to distinct states. Our discrete C-space, C, is then simply a subset of T. The robot is allowed to move around in this subset from cube to cube, but may not enter it's complement, $T - C$, which we denote as C^c.

Since the space that we will do our planning in is discrete, it is not clear what should be meant by a continuous path in it. We say that two cubes are *adjacent* if they intersect in a face (eq. if their corresponding center points differ only by a unit change in one of their coordinates). For example, in two dimensions a square has 4 squares adjacent to it, and in three dimensions a cube has 6 cubes adjacent to it. By a *path* in C from p to q we mean a sequence of cubes in C, $p = p_1, p_2, ..., p_r = q$ such that any pair of consecutive points in this sequence are adjacent. What this means for the robot in the real world is that it is allowed to move in at most one direction at a time, and only in unit increments.

A *boundary cube* of C is a cube $p \in C$ such that there is a cube lying in C^c, and the two cubes intersect in an edge or a face.

Corresponding to C and C^c are the underlying spaces, $\bar{C}$ and $\bar{C}^c$ which are subsets of R^n. By a hyperface of $\bar{C}$ or $\bar{C}^c$ we mean a hyperface of a cube in C or C^c. The set of boundary points of $\bar{C}$ is equal to the set of boundary points of $\bar{C}^c$, and we will denote this set by ∂C, discarding the bar over the C to keep the notion simple.

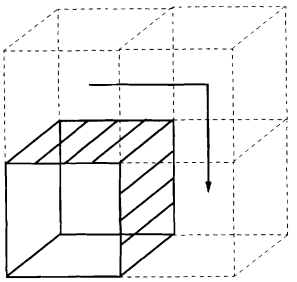


Figure 4: The robot may move from one face of ∂C to another, but it needs to go through "edge" cubes.

6 The Algorithm Makepath

In this section we give our algorithm *Makepath*, which constructs a path between two cubes in C if such a path exits, or indicates that no such path is possible. Our fundamental assumption of local sensing is: **if the robot occupies a cube p, in C, then for any cube (not just the adjacent ones) intersecting p, the robot can tell if that cube lies in C or C^c.** Our basic premise is that when doing this sort of very local planning, the robot should navigate on the boundary of the configuration space, and thus the question that needs to be answered is how efficiently this can be done.

Step I Suppose that we fix C and C^c and we are given a pair of states *start* and *goal* in C. The first step is to "digitize" the straight line in R^n that connects the center points of *start* and *goal*, i.e. compute a path l in T from *start* to *goal*. The calculation of l is an internal operation that does not require any global knowledge except the coordinates of *goal*.

Step II The robot starts at *start* and moves along l, from one cube to the next, towards *goal*, until it finds that the next point on the line lies in C^c, i.e. the robot has bumped into an obstacle. The robot then occupies a cube *entry*.

Step III This step requires the algorithm $Surfacesearch$, which searches the surface for an exit point. Roughly, *Surfacesearch* is an on-line depth-first search through all of the boundary cubes of C, starting at *entry*. If the robot enters a cube where all adjacent cubes have been visited, then it performs an internal breadth-first search, looking for a boundary cube that it has not visited. Once it determines the location of such a cube, it then backtracks to it, through cubes that it has already entered. Upon arriving at this cube, it then returns to step II.

7 Searching the Surface

The robot's state is the cube *entry*, and it now navigates in the space of boundary cubes, looking for an exit cube. It is always on the look out for a cube that lies on l that is closer to *goal* than is *entry*. We will call such cubes *exit* cubes. If we consider the space of boundary cubes of a connected piece of ∂C, then it is path connected with respect to our above definition of path. We would like to find an exit cube, which is actually a cube of C that intersects ∂C in an entire face. Unfortunately, the set of cubes that intersects a component ∂C in an entire face is not connected. We need to add in the "edge" cubes, in order to get from one such cube to another(see figure 4).

The robot then begins an on-line depth-first search through the space of boundary cubes, starting at *entry*. As it moves, it creates a record, *Cubes*, which contains a list of all the cubes that the robot has sensed, and which ones it has actually occupied. The robot moves from one cube to the next available boundary cube if one is available. If several choices are available, for now we may assume that it chooses one arbitrarily.

The robot may get into a situation where no boundary cubes are available for it to move to. In this case it looks at its record, *Cubes*, and does an internal breadth-first search for the nearest boundary cube that it has not entered. So it travels back and resumes its on-line search. From a geometric viewpoint, the robot is covering a component of ∂C.

8 Runtime Issues

To see how many moves the robot may have to make, we count the number of moves it makes during the periods when it is in the searching phase and the number of moves it makes during the backtracking phases.

During the search phase a cube is never entered twice. Therefore the maximum number of move is equal to the number of boundary cubes.

During backtracking, as mentioned above, the robot need never backtrack farther than the length of the path it has generated, since it can simply travel back on the same path that it came on. Since this is always a choice, that means that an upper bound for the backtracking is equal to at most the number of boundary cubes. But most of the time one would expect the robot to find a much shorter way back.

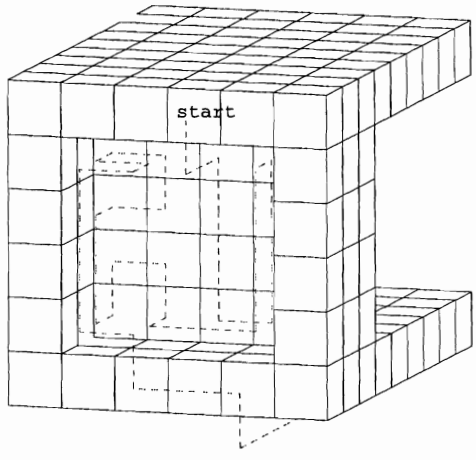


Figure 5: Here the robot searches a surface (the dotted line indicates the path). Using a gradient flow, it gets stuck in a pit.

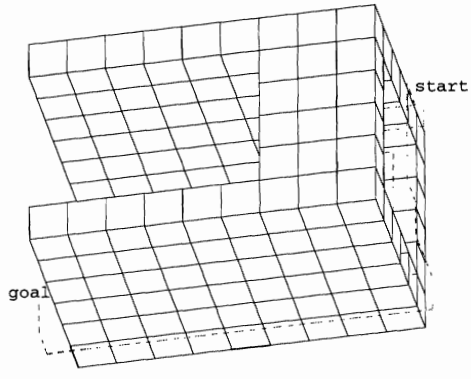


Figure 6: After the robot climbs out of the pit, it is lucky and can head straight to the goal. In this case, it never has to do any backtracking.

Thus, while not traveling on l, the robot need never move more than twice the number of boundary cubes. Adding to this the number of moves it makes on the straight line from start to goal, and denoting the number of boundary cubes as S, gives a total number of $2S + d$ as an upper bound for the number of moves it needs to make. It would seem that a lower bound would be close to $S + d$, given the above framework, since the robot might have to search the whole surface. It certainly seems that in the worst case the robot would have to search almost the entire surface, for essentially the reason given in figure 1. Fortunately there a two facts that can possibly be exploited to avoid this in an implementation.

First, the robot may have sensors, that, at a fixed time see a part of the surface that is much larger than just a "quantum unit", as we have considered here. For example, in the planar piano movers problem, the boundary of the configuration space is two dimensional, but if a robot with tactile sensors makes a one dimensional motion, e.g. slides along while touching a wall, then it retrieves two dimensional data, since once it knows where the wall is, it can internally reconstruct a large piece of the C-space boundary. Thus one should take care to distinguish between sensor based planning for mechanical systems, as opposed to sensor based planning for a point in an arbitrary space.

A second possible way to speed up the search is to use potential fields. The authors wrote several simulators, and in figure 5 and figure 6 are two plots from one run of a simulator. These plots show an obstacle surface, (the cubes), and a path from a given start to a goal (the dotted line). Here we have chosen the start and the goal to be near the surface. In this example the robot finds the goal without ever getting stuck. In figure 5 we see that the robot is on the back of the object and that it walks into a "pit". But after it gets out of the pit, in figure 6, we see that it travels directly to the goal. The reason for this is that in this implementation we chose not to walk randomly on the surface, but roughly by moving in the direction of the gradient of the distance function to the goal. If the gradient vanishes, then we do choose our next position randomly.

References

[1] A. Blum, P. Raghavan and B. Scheiber, *Navigating in Unfamiliar Geometric Terrain*, SIAM J. Comput., 26(1997), pp. 110-137.

[2] E. Cheung and V. Lumelsky, *Motion Planning for a Whole sensitive Robot Arm Manipulator*," Proc. of the IEEE International Conference on Robotics and Automation, Cincinnati, OH, 1990, pp. 344-349.

[3] H. Choset and J. Burdick, *Sensor Based Planning for a Planar Rod Robot: Incremental Construction of the Planar Rod-HGVG* , in Proceedings IEEE/ICRA, New Mexico, USA, 1997.

[4] J. Cox and C.K. Yap, *On-line Motion Planning: Case of a Planar Rod*, Annals of Mathematics and Artificial Intelligence, 3(1991), pp. 1-20.

[5] B. R. Donald, *Motion Planning with Six Degrees of Freedom*, Report No. MIT AI-TR 791, MIT, Artificial Intelligence Laboratory, 1984.

[6] B. R. Donald, *A Search Algorithm for Motion Planning with Six Degrees of Freedom*, Artificial Intelligence, 31, 1987, pp. 295-353.

[7] I. Kamon, E. Rimon and E. Rivlin, *A new range-sensor based globally convergent navigation algorithm for mobile robots*,IEEE Conference on Robotics and Automation(1996), pp. 429-435.

[8] K. N. Kutulakos, C. R. Dyer, and V. J. Lumelsky, *Provable Strategies for Vision-Guided Exploration in Three Dimensions* , Proc. 1994 IEEE Int. Conf. Robotics and Automation, 1994, pages 1365-1372.

[9] K. Kutalakos, V. Lumelsky and C. Dyer, *Vision-Guided Exploration: A Step Toward General Motion Planning in Three Dimensions*, IEEE Conference on Robotics and Automation(1993), pp. 289-296.

[10] K. Kutalakos, V. Lumelsky and C. Dyer, *Vision-Guided Exploration: A Step Toward General Motion Planning in Three Dimensions*, University of Wisconsin Computer Sciences Department Technical Report 1111(1993).

[11] J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publish ers, Boston, MA, 1991.

[12] J. Lengyel, M. Reichart, B. Donald, D. Greenberg, *Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware*, Proceedings of SIGGRAPH'90, Dallas, Tx, 1990, pp. 327-335.

[13] V.J. Lumelsky , *Algorithmic and Complexity Issues of Robot Motion in an Uncertain Enviorment*, Journal of Complexity(3),1987, pp. 146-182.

[14] V.J. Lumelsky and A.A. Stepanov, *Dynamic path planning for a mobile automaton with limited information on the environment*, IEEE Trans. Automatic Control, AC-31(1986), pp. 1058-1063.

[15] V.J. Lumelsky and K. Sun, *A Unified Methodology for motion planning with uncertainty for 2d and 3d two-link robot arm manipulators*, Int. J. of Robotics Research, vol. 9, no. 5, pp.89-104, 1990.

[16] C.H. Papadimitriou and M. Yannakakis, *Shortest paths without a map*, in Proc. 16th International Colloquim on Automata, Languages, and Programming, Springer-Verlag, Berlin, 1989, pp. 610-620.

[17] K. Sun and V. Lumelsky, *Path Planning among unknown obstacles: the case of a three dimensional cartesian arm*, IEEE Trans. on Robotics and Automation, vol. 8, no. 6, pp. 776-786, 1992.