OBSTACLE AVOIDANCE USING AN OCTREE IN THE CONFIGURATION SPACE OF A MANIPULATOR.

Bernard FAVERJON

INRIA
Domaine de Voluceau B.P. 105
78153 LE CHESNAY CEDEX, FRANCE

ABSTRACT

An automatic system for planning safe trajectories for a computer controlled manipulator among obstacles is a key component of robot assembly operations.
This paper describes an algorithm transforming cartesian obstacles into obstacles in the space of the first three joints of a manipulator with six revolute joints (e.g. a ACMA-CRIBIER V80), and giving a hierarchical description of the free space by mean of an octree.
Such a description is very useful in testing for collision between the arm of the manipulator and obstacles since it is represented by a point in this space.

1 INTRODUCTION

We first describe different types of algorithms that can be used in obstacle avoidance problems and the usefulness of the representation of free space we propose for solve these problems.
Section 2 describes the transformation algorithm.
Section 3 describes some examples of algorithms using this representation.

Obstacle avoidance algorithms

The simplest algorithm for planning free paths amongst obstacles uses the generate and test method. A simple path from start to goal is hypothesized and is tested for potential collisions. If a collision is detected, a new path is proposed using information about this collision. This is repeated until no collisions are detected along the path.
In the case of a manipulator such an algorithm can be described in three steps :

1-calculate the volume swept out by the manipulator along the proposed path
2-determine the overlap between the swept volume and the obstacles
3-propose a new path

Such an algorithm presents several difficulties and drawbacks.
First, calculating the volume swept out by a manipulator with revolute joints is hard. The output we can expect is a description of this volume with a set of many simple surfaces. A similar description of the obstacles can be provided, and we are faced with the problem of determining the overlap of such volumes which is known to be difficult and time expensive.
Another important problem lies in the relationship between the second and the third steps. The information we can expect from the second step is only local (i.e. it concerns only a part of the path and a part of the manipulator). As the manipulator consists in several parts linked together, it is difficult to find good heuristics to modify the paths. But even with good heuristics, the local character of this method makes impossible great changes in the path. So the proposed paths are generally not short relative to some criteria we would like to minimize (e.g. execution time of the path by the manipulator).

For these reasons, another type of algorithm has been used independently for manipulator obstacle avoidance by Udupa (1) and Lozano-Perez (2-3).
In this method, the goal is to simplify the description of the moving object while transforming the obstacles such that an overlap between the new object and the new obstacles is equivalent to an overlap between the original ones. The first two steps of the algorithm described above are then simplified.

Udupa's work concerned the Standford arm. It used the fact that this manipulator is composed of two linked objects, one of them (the boom) being much larger than the other (the forearm).
The manipulator is decomposed in these two parts which are then approximated by cylinders. An abstraction space is constructed by computing the forbidden regions for the boom tip in spherical coordinates.
The volume swept out by the boom in this space is represented by a curve and the above algorithm can be applied easily. When a collision free path for the boom has been found, a path for the forearm is searched along the boom tip locus using heuristics based on the

-the body which can rotate about a vertical axis (joint q1)
-the upperarm attached to the body by an horizontal axis intersecting the first one (joint q2)
-the forearm attached to the upperarm by an horizontal axis (joint q3)
-the hand attached to the wrist by three intersecting axis (joints q4,q5,q6)
we call arm of the manipulator the union of the body,the upperarm and the forearm.

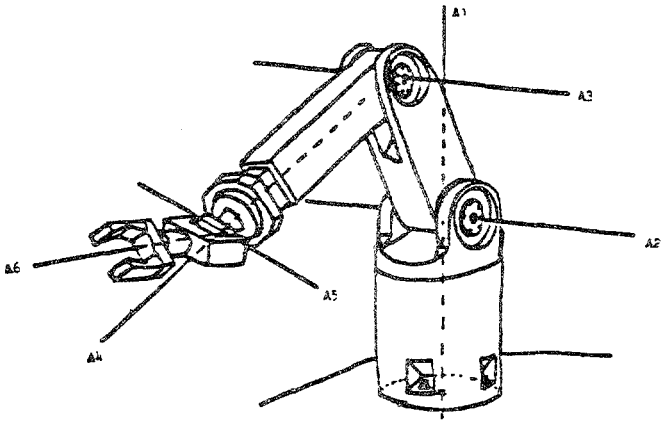

Figure 1 : the ACMA-CRIBIER V80 manipulator.

We now define some notations used in the following.

We denote by E the three dimensionnal cartesian space.

Q=Q1XQ2XQ3 is the set of values that joints (q1,q2,q3)=q can take according to the constructional constraints of the manipulator. Qi, i=1,3 is an interval contained into -pi,+pi .

Aldesignes the body, A2 the upperm, A3 the forem and A=A1UA2UA3 the arm.

A cartesian obstacle will be denoted by O and its corresponding transformed obstacle by T(O). T(O) is defined as the set of points q of Q such that A(q) overlaps with O.

2-2 simplifing the problem

The real volumes A1,A2,A3 are of course complicated. Since we have seen that a rough approximation of the arm suffises in most of the cases, we replace them by bounding volumes made of a cylinder ended by hemispheres, all of the same radius R. Let Si, i=1 to 3 be, the line segments such that Ai is the set of points distant of less than R from Si and S=S1uS2uS3.
It can be seen easily that T(O) can be defined as the set of points q of Q such that S(q) overlaps the grown obstacle G(O). G(O) is defined as the set of points of E distant of less than R from O.
Before we give an important lemma which is the basis of the practical algorithm, notice that the representation of a cartesian position in the joints space is not unique. If (q1,q2,q3) is a representation then (q1+pi,pi-q2,-q3) is also a representation of the same position. For this reason, we only consider a part of T(O) in the following and the other can be deduced from it using the relation above.

Lemma : If the cartesian obstacle O is convex, the transformed obstacle T(O)can be written :
$T(O) = U_{q1<I1} U_{q2<I2(q1)} \; q1 \times q2 \times I3(q1,q2)$

where Ii is an interval included in Qi.

Proof : notice first that segments Si are contained in a vertical plane defined by joint q1. So we can write $T(O) = U_{q1<I1} T(Oq1)$,

where Oq1 is the overlap between O and such a plane. I1 is the set included in Q1 such that T(Oq1) is not empty, that is there exists q2 and q3 such that S(q) overlaps with G(Oq1). Let B1 be the common end of S1 and S2, and l2 and l3 the respective length of S2 and S3. If the distance of B1 to G(Oq1) is greater than l2+l3 , then T(Oq1) is empty. We conclude that I1 can be defined as the set of angle q1 such that the sphere centered in B1 of radius R overlaps with G(Oq1). As this sphere and G(O) are convex, it can be seen easily that I1 is also convex and so is an interval.

We now examine the set T(Oq1). It is clear that it can be written :
T(Oq1) = q1 X TP(Oq1) where TP is the transformation of a cartesian obstacle contained into a plan containing axis z into the joints space (q2,q3).
Let B2 be the common end of S2 and S3. If the distance of B2 to G(Oq1) is greater than l3, then the overlap between S3 and G(Oq1) is clearly empty and we can define I2 as the set of angles q2 such that S2 overlaps with G(Oq1) or this distance is less than l3. (figure 2). If l3 is greater than l2 the first condition is implied by the second.

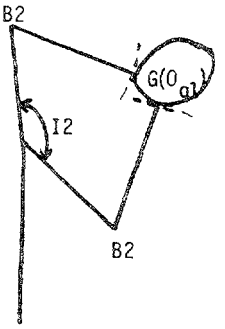

Figure 2 : Definition of the set I2.

-the body which can rotate about a vertical axis (joint q1)
-the upperarm attached to the body by an horizontal axis intersecting the first one (joint q2)
-the forearm attached to the upperarm by an horizontal axis (joint q3)
-the hand attached to the wrist by three intersecting axis (joints q4,q5,q6)
we call arm of the manipulator the union of the body,the upperarm and the forearm.

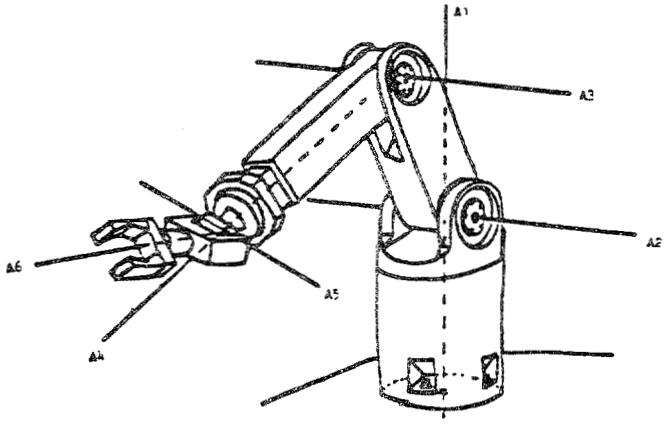

Figure 1 : the ACMA-CRIBIER V80 manipulator.

We now define some notations used in the following.

We denote by E the three dimensionnal cartesian space.

Q=Q1XQ2XQ3 is the set of values that joints (q1,q2,q3)=q can take according to the constructional constraints of the manipulator. Qi, i=1,3 is an interval contained into -pi,+pi .

Aldesignes the body, A2 the upperm, A3 the forem and A=A1UA2UA3 the arm.

A cartesian obstacle will be denoted by O and its corresponding transformed obstacle by T(O). T(O) is defined as the set of points q of Q such that A(q) overlaps with O.

2-2 simplifing the problem

The real volumes A1,A2,A3 are of course complicated. Since we have seen that a rough approximation of the arm suffises in most of the cases, we replace them by bounding volumes made of a cylinder ended by hemispheres, all of the same radius R. Let Si, i=1 to 3 be, the line segments such that Ai is the set of points distant of less than R from Si and S=S1uS2uS3.
It can be seen easily that T(O) can be defined as the set of points q of Q such that S(q) overlaps the grown obstacle G(O). G(O) is defined as the set of points of E distant of less than R from O.
Before we give an important lemma which is the basis of the practical algorithm, notice that the representation of a cartesian position in the joints space is not unique. If (q1,q2,q3) is a representation then (q1+pi,pi-q2,-q3) is also a representation of the same position. For this reason, we only consider a part of T(O) in the following and the other can be deduced from it using the relation above.

Lemma : If the cartesian obstacle O is convex, the transformed obstacle T(O)can be written :
$T(O) = U_{q1<I1}\ U_{q2<I2(q1)}\ q1 \times q2 \times I3(q1,q2)$

where Ii is an interval included in Qi.

Proof : notice first that segments Si are contained in a vertical plane defined by joint q1. So we can write $T(O) = U_{q1<I1}\ T(Oq1)$,

where Oq1 is the overlap between O and such a plane. I1 is the set included in Q1 such that T(Oq1) is not empty, that is there exists q2 and q3 such that S(q) overlaps with G(Oq1). Let B1 be the common end of S1 and S2, and l2 and l3 the respective length of S2 and S3. If the distance of B1 to G(Oq1) is greater than l2+l3 , then T(Oq1) is empty. We conclude that I1 can be defined as the set of angle q1 such that the sphere centered in B1 of radius R overlaps with G(Oq1). As this sphere and G(O) are convex, it can be seen easily that I1 is also convex and so is an interval.

We now examine the set T(Oq1). It is clear that it can be written :
T(Oq1) = q1 X TP(Oq1) where TP is the transformation of a cartesian obstacle contained into a plan containing axis z into the joints space (q2,q3).
Let B2 be the common end of S2 and S3. If the distance of B2 to G(Oq1) is greater than l3, then the overlap between S3 and G(Oq1) is clearly empty and we can define I2 as the set of angles q2 such that S2 overlaps with G(Oq1) or this distance is less than l3. (figure 2). If l3 is greater than l2 the first condition is implied by the second.

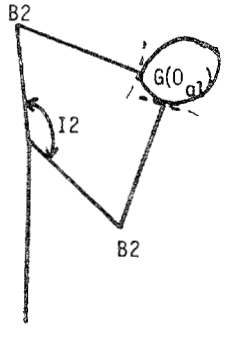

Figure 2 : Definition of the set I2.

We now show that I2 defined with only the second condition is convex. Let q21 and q22 be two elements of I2. If they are both such that S2 overlaps with G(Oql) which is convex, it is easy to see that any q2 between q21 and q22 is in I2. Else they correspond to two positions of B2, B21 and B22, and we suppose that for any q2 betweem them S2 doesn't overlap with G(Oql). (if not we cut the segment q21,q22 in three parts which fall in one of the two cases). There exist two points M1 and M2 of G(Oql) such that D(B21,M1) and D(B22,M2) are both less than l3 and it can be seen that in those conditions the distance from B2 to the segment M1,M2 can only have a minimum when q2 describes the segment q21,q22 (figure 3). From the convexity of G(Oql), we deduce that q2 is in I2, which is then convex and an interval contained in Q2.

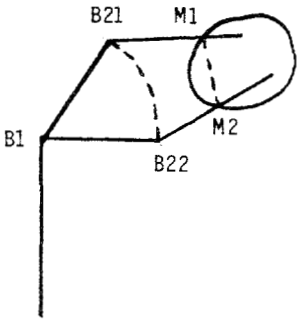

Figure 3

If q2 is such that S2 overlaps with G(Oql), it is clear that the whole interval Q3 must be forbidden and so I3(q1,q2) = Q3.
Else, the forbidden set for q3 is such that S3 overlaps G(Oql). Again, it is easy to see using the convexity of G(Oql) that I3 is convex and an interval included in Q3 (figure 4). That proves the lemma.

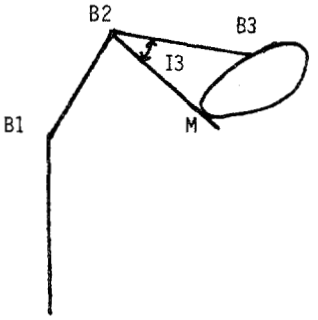

Figure 4 : Definition of the set I3.

2-3 The description of obstacles.

We can now choose a description of the obstacles that simplifies the transformation. First, we have to describe them as unions of convex sets. Then, we have seen that the transformation involves the determination of intervals using the cartesian distance from a point to the grown obstacle or the search of tangent points between a segment and the grown obstacle. (we define M as a tangent point if the overlap between the segment and the grown obstacle is equal to M and is not an extremity of the segment). All these operations are quite simple if the grown obstacle is a polyhedron or a sphere and we will use such approximations of the grown obstacles.

We can expect two different sources for the description of the world. We can use geometric models of the objects given by a CAD system or the information given by a 3-D sensor. In the last case, the information can be condensed into a polyhedron and then a hierarchical structure, the prism-tree described by Faugeras and Ponce (4), by means of prisms. The CAD system provides a similar description, but uses several basic objects as cuboids,cones,cylinders or spheres.
Such basic objects will be the input of the growing algorithm that generates polyhedra and spheres as output. A basic object is defined by its type and parameters of size and position. A polyhedron by a set of vertices and a set of edges. For a given type of basic object the set of edges describing the shape of the output polyhedron is always the same, and only the coordinates of the vertices depends on the parameters of this object. Of course, the grown obstacle of a sphere of radius r is a sphere of radius r+R.
So, the growing operation is quite simple and fast.

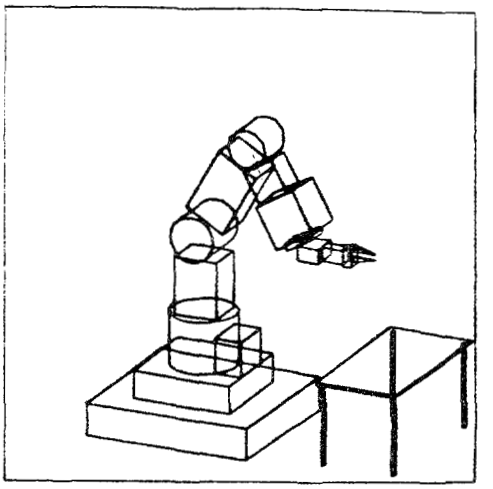

Figure 5 : a scene described by the CAD system.

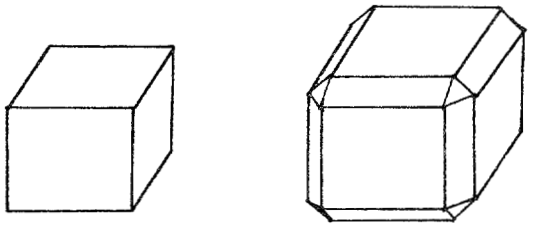

Figure 6 : a cuboid and the grown obstacle.

2-4 the transforming algorithm.

We describe here in details the case of a polyhedron as grown obstacle. The case of a sphere is easier and very similar.
We must keep in mind that our purpose is to build an octree in the joint space of the arm. An octree is a tree of degree eight which describes hierarchically the space contained into a cuboid that forms the root. The sons of a node are the eight cuboids obtained in cutting the father by planes parallel to the faces and containing its center. The nodes can be labelled as Full,Empty,or Mixed depending on whether they are entirely in an obstacle, out of all obstacles, or partly in an obstacle. Only the Mixed nodes are divided until the minimum size for a cuboid is reached. A cuboid of minimum size is called a voxel.
In our case, the root is the set Q, and we divide each coordinate into 64 parts. We obtain a 3-D image of size 64X64X64 that we have to fill using the transforming algorithm. We denote dq1,dq2,dq3 the increments on the various angles.
The transforming algorithm uses the results of the lemma and can be described as follows :
1-cut the polyhedron into slices by planes containing the z axis and a vertex;
2-for all the slices do :
  If the distance between B1 and the slice is less than I2+I3 then
    Cut the slice into sub-slices of thickness dq1;
    compute TP(Oq1);
  Else next slice;
end;

G(Oq1) is a convex polygon. I2(q1) can be computed as the union of the same set for each edge denoted I2n. For a given q2 in I2, I3(q1,q2) is the union of the same set denote I3n for the edges n such that q2 is in I2n. Notice that the extremities of I3 (when it is different from Q3) correspond to points on two of these edges that remain on these edges until the set of edges n concerned with q2 changes. The part of I2(q1) in which S2 overlaps G(Oq1) (that is I3 ∩ Q3) can be easily computed and is denoted by I2o(q1).

So, the computation of TP(Oq1) can be described as follows :
1-determine all I2n and their union I2(q2) and I2o(q1);
2-determin e the edges concerned with q2 = min I2(q1) and the two particular edges;
3-compute the points of contact on this two edges and the corresponding points in Q3, q3m and q3M; I3 = q3m,q3M ;
4-fill all voxels corresponding to (q1,q2,q3) with q3 in I3;
5-add dq2 to q2;
  If q2 is in I2o(q1) then I3 = Q3 go to step 4;
  if the concerned edges are not changed, go to step 3;
  if the set of concerned edges is empty then return;
  else determine the two new particular edges and go to step 3;
end;

2-5 Building the octree.

When the filling of the basic 3-D image is finished, we have to build up the octree. As we want to use the neighbor information in the following we choose to number the nodes as if the octree was full up. At each level l, the voxels are defined by three integer coordinates n1,n2,n3 between 0 and 2^l-1. The corresponding node in the octree will be denoted (l,n1,n2,n3). His father is then given by (l-1,n1/2,n2/2,n3/2) and his eight sons by (l+1,n'1,n'2,n'3) with n'i = 2ni or 2ni+1.
With this representation, only two bits are necessary for each cell. Their meaning is the following :
  1 1 = the node is entirely full.
  1 0 = the node is mixed.
  0 1 = the node is empty and is a leaf (i.e. iTs father is not empty)
  0 0 = the cell is empty but is not in the octree. (it is a descendant of an empty node.

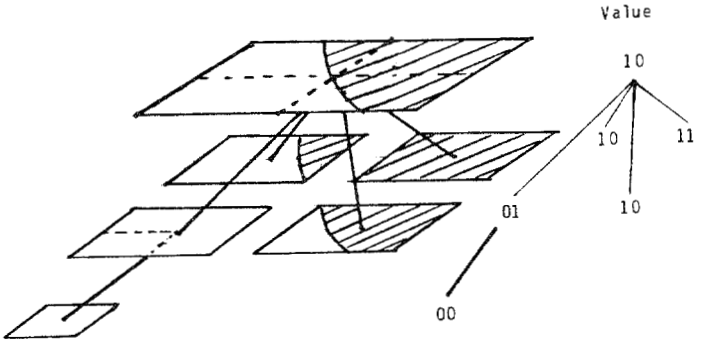

Figure 7 : Labels of the nodes in the octree

the building algorithm is then :

```
for l = 5 to 0 do
  for n1 = 0 to 2^l-1 do
    for n2 = 0 to 2^l-1 do
      for n3 = 0 to 2^l-1 do
        If the 8 sons of node (l,n1,n2,n3)
        are full, set it to 1 1;
        if the 8 sons of node (l,n1,n2,n3)
        are empty, set it to 0 0;
        else set it to 1 0 and its empty sons
        to 0 1;
      end;
    end;
  end;
end.
```

Of course, this algorithm is not optimal as that described by Samet (5-6) for quadtrees but finding the neighbors of a node in the octree is much faster with our coding. (see section 3-2).

3-ALGORITHMS USING THIS REPRESENTATION.

We describe here two algorithms using this representation of the manipulator environment. The first one tests for overlap between the arm and obstacles along a cartesian path for the wrist. The second one searches for a short path for the arm in the graph of the neighbors of the octree relative to a pre-specified criterion.

3-1 testing a cartesian path.

Such an algorithm can be used in the first step of the find path problem when the search is led by the configuration of obstacles around the end effector.
The first step is to find an approximation of the cartesian path made of successive line segment motions in the joints space of the arm. Then we test if each of these segments is contained in the free space described by the octree. Let qi and qf be the extremities of such a segment. Starting with the root of the octree, we first follow a branch until we find a full or an empty node containing qi or qf. Let Ni and Nf be these nodes. If one of them is full we return False. If Ni is equal to Nf we return True. Else, we compute the intersections qi' and qf' between the segment and the surface of nodes Ni and Nf and we call recursively for the test of segment (qi',qf'). In order to insure the new points won't be in the same cells as the old ones, these are grown by a small distance.
This algorithm is very efficient to test for collision between the whole arm of the manipulator and obstacles.

3-2 searching a short path for the arm.

Such an algorithm can be used in the second step of a finding free path problem when looking for large motion of the arm that minimizes a pre-specified criteria. It uses the graph of neigbors of the octree and is derived from the well known A* algorithm. This algorithm allows the use of heuristic information. Initially, the start node is placed on a list of candidate nodes for examination (the OPEN list). At each step of the algorithm ,the node with minimum total path cost estimate (i.e. actual cost of reaching the node from the start plus an estimate of the cost to travel from the node to the goal node) is moved onto a CLOSE list and its neigbors are placed on the OPEN list. The search ends when the goal node is moved onto the CLOSE list. The algorithm finds an optimal path when the estimate cost is a lower bound of the true cost.

We first consider the neighbor-finding algorithm.
Let (l,n1,n2,n3) be a node of the octree. We want to find all its empty neighbors in the octree. The neighbors of a node are the nodes that share a face with it. At level l, there exist six cells neighboring this node, but they may not be in the octree. Consider one of these cells. If it is full there cannot be neighbors in this direction. If it is empty and a leaf (type 01), it is the only neighbor in this direction. If it is empty and not a leaf (type 00), one of its ancestors is the only neighbor in this direction. Finaly, if it is mixed, we call recursively for the neighbors of its four sons sharing the concerned face in this direction.

We now examine different choices of criteria.
The criterion we want to minimize is the execution time of the path by the manipulator. As we work in the joints space of the arm the path will be defined by a succession of line segments in this space. The time for such a path includes two terms : one for the motion at constant speed and one for the changes of speed. For a segment, the first one is proportional to the maximum angle of rotation on the different axis and the second one to the maximum change of rotation speed on the different axis. This second term is difficult to compute because we don't only need the current node but also the previous one. If we only take into account the first term, the results are not very good, because the criterion is constant on cuboids and the choice between the nodes in the same faces of such surfaces is quite arbitrary. In order to take into account the changes in the direction of speed, it is in fact better to choose the cartesian distance wich gives results similar to the complete criterion but is much easier to compute.
Another problem is the choice of end points for segments. The simplest is to choose the centers

of the cells, but one can see that large cells are then disadvantaged. A better solution is to choose the point of the cell that minimize the cartesian distance to the actual point and the goal, but of course it is much longer. Another solution is to correct the simple criterion by adding a cost depending on the size of the cell. When an optimal list of cells has been found, intermediate points are searched in their union such that the total distance is minimized. These points are vertices or on edges of the cells.

Until now we have searched for an optimal path for the arm only. The following step is to find a correct orientation for the hand along the path for the wrist. This is of course easier if the wrist is not too close to the obstacles. So, we had better to search for a path far from obstacles than a shorter one close to obstacles for the arm. In order to do that we can increase the additionnal cost on the size of the cells. Figures 8 and 9 shows two trajectories with the same start and goal positions. The first one has been found with a low cost one size of cells and the second with a heavy cost.

Speeding up the search.

One easy way to speed the search is to use a heuristic. In the case we use the cartesian distance in joints space as criteria, a lower bound of the cost of the remaining path from the current node to the goal is of course the cartesian distance between these two points and we will use this heuristic.
But even with a heuristic, the search may remain long. To reduce the time we can now use the hierarchy of the octree. Instead of giving a heavy cost to small cells, the idea is simply not to consider them in the search. Let lS and lG be the level of cells containing the start and goal points and l the maximum level at which we want to search for a path. The real level used in the search is max(l,lS,lG) and is incremented by 1 if no path can be found. The time of the search is about ten times smaller for l = 4 than for l = 6.
The problem in this method is that the level is limited by lS and lG. A solution is to use a modified version of the A* algorithm in which the search is done from both the start and goal nodes at the same time. In this case, we use an OPEN and a CLOSE list for both sides. When a node is in both CLOSE lists the search ends and the optimal path is the union of the optimal paths reaching the start (resp. the goal) node to this node. We can then start the search at level lS (resp. lG) at start (resp. goal) side. As soon as a node of smaller level is moved to the CLOSE list we can limit the search to this level, if it is greater than l and to l otherwise. If no path can be found l is incremented by 1 and the search done again.

CONCLUSION.

By transforming the cartesian obstacles into a hierarchical structure in the joints space of the arm of a manipulator with revolute joints we are able to test and search for paths in a cluttered environment very quickly. This method is useful when many motions have to be done in the same environment. The construction of the representation of free space is done in less than 15 seconds on a mini-computer Perkin-Elmer 3244 for the environment of figures 8 and 9. (the computation of a transformed obstacle is about proportional to the solid angle of the cartesian obstacle seen from point B1). Collision-free paths are found in about 5 seconds using the whole octree and less than a second when limiting the level of search.

REFERENCES.

(1) Shriram M. UDUPA
"collision detection and avoidance in computer controlled manipulators."
proceedings of IJCAI-5 MIT Cambridge,Ma.,Aug 77

(2) Thomas LOZANO-PEREZ
"an algorithm for planning collision-free paths amongst polyhedral obstacles."
IBM Research report. RC 7171 June 78

(3) Thomas LOZANO-PEREZ
"automatic planning of manipulator transfer movements"
IEEE Trans. On Systems,Man and Cybernetics (681-698) SMC-11

(4) Olivier D. FAUGERAS and Jean PONCE
"prism-trees : a hierachical representation of 3-D objects"
IJCAI-83, Karlsruhe,Germany Aug. 83

(5) Hanan SAMET
"Region representation : quadtrees from binary arrays"
Computer graphics and IP 13 (88-93) 1980

(6) Hanan SAMET
"Neighbor finding techniques for images represented by Quadtrees."
Computer graphics and IP 18 (37-57) 1982

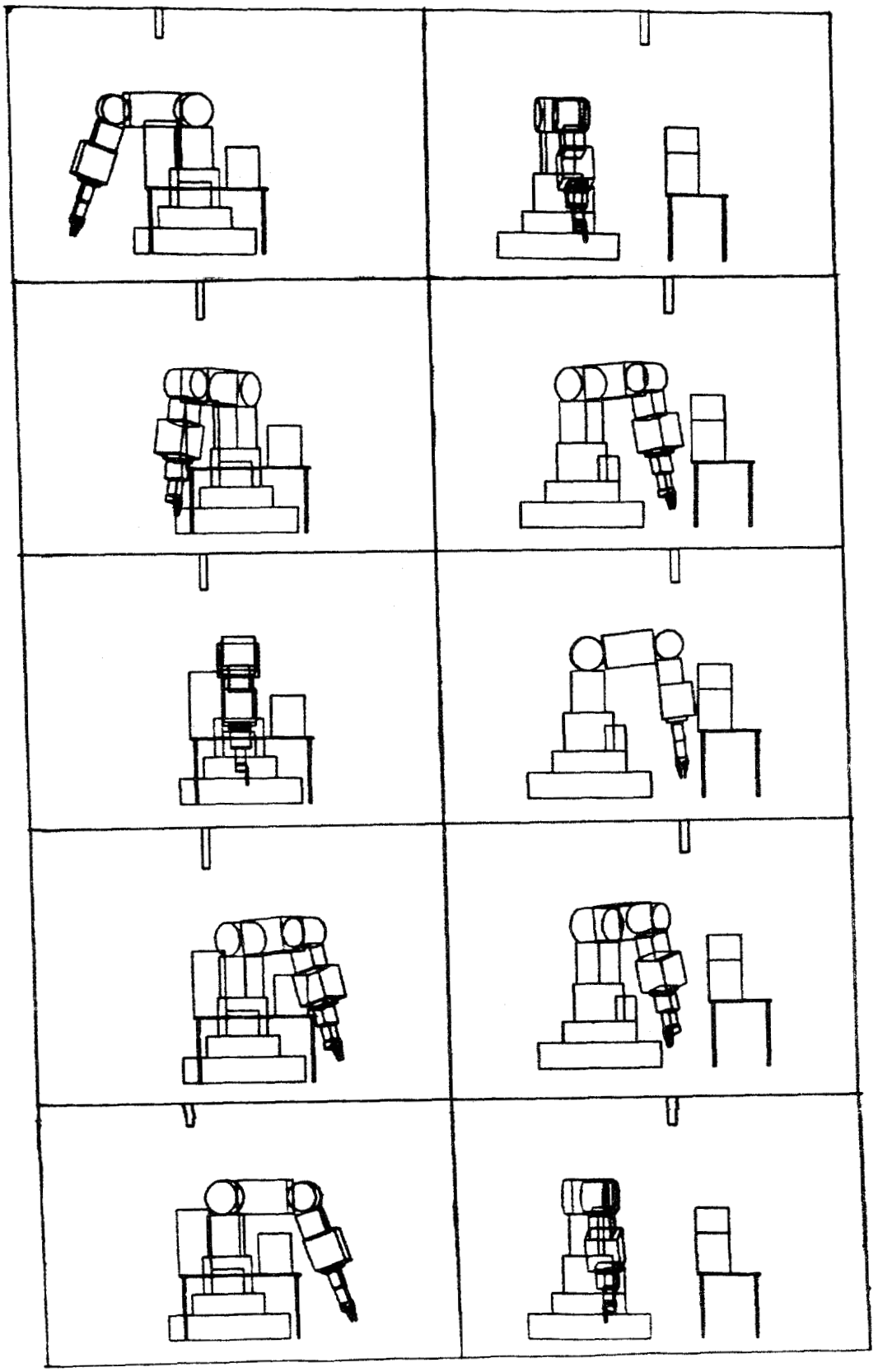

Figure 8 : A path with a low cost on small cells.

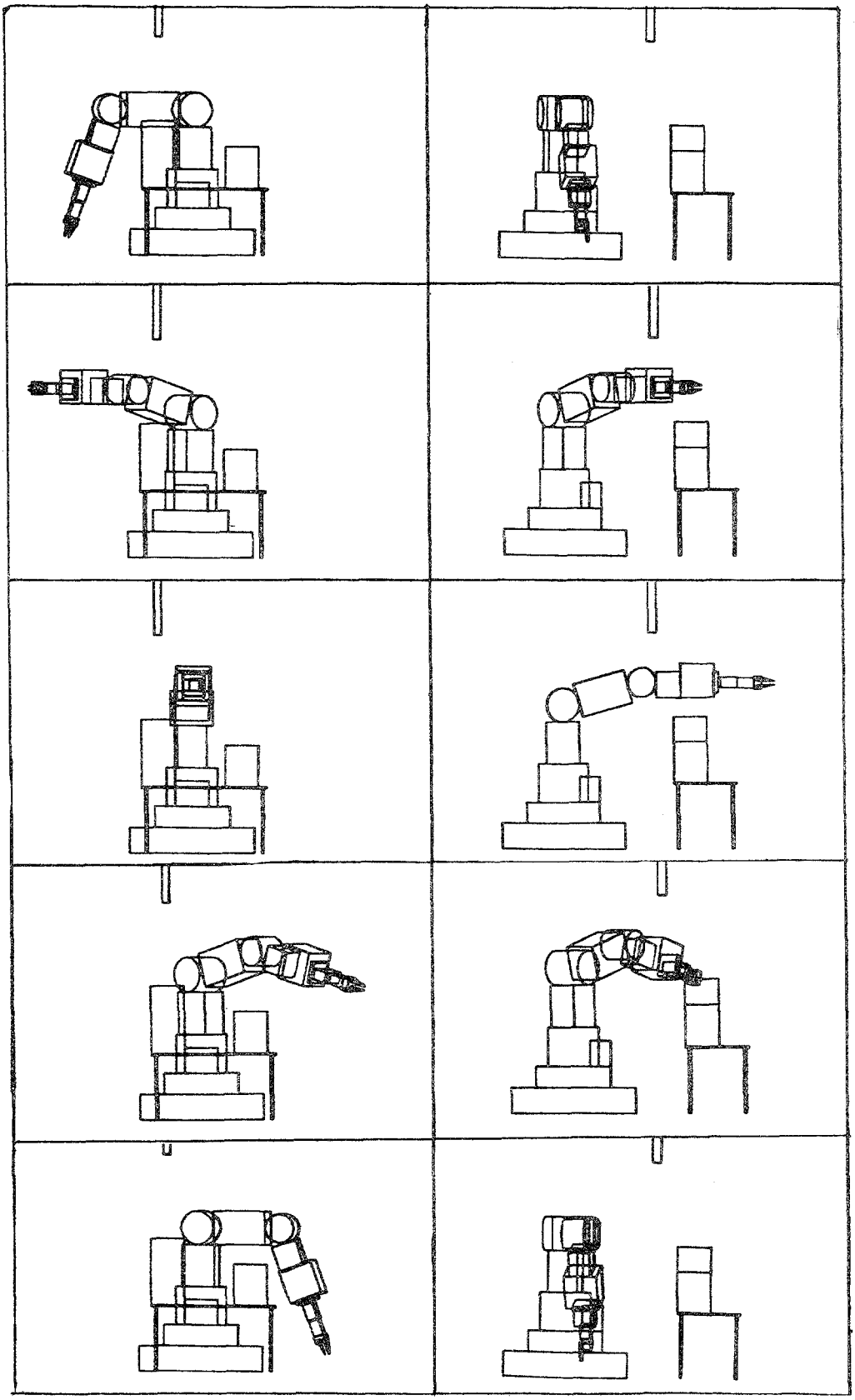

Figure 9 : A path with a heavy cost on small cells.