Toward Efficient Path-Planning for Articulated Robots

G. Gini* R. Massa† and R. Negretti
Department of Electronics and Information
Politecnico di Milano, Italy

Received January 18, 1993; revised November 16, 1993;
accepted April 2, 1994

Real-time issues are becoming more and more important in robot programming. When a 6-dof manipulator is used, planning obstacle-avoiding paths is a time-consuming activity, usually done in simulation. We present the geometric models and the reasoning techniques we have implemented while realizing a gross motion planner for a manipulator with six revolute joints. First, construction of a problem-oriented representation of the robot working space is explained. Then, the actual trajectory research carried out in our C-space representation is described. The whole C-space is not calculated; instead, a sequential strategy is used to determine the C-space only for the first two links. Our approximation of the obstacles, which occupy fixed and known positions, greatly speeds the computation, allowing us to reduce the problem to planar geometric reasoning. The work is not limited to theoretical studies or simulations; experiments have been run very thoroughly, with various tests, on a PUMA robot to assess the real efficiency and usability of our software. The method applies to robots in a fixed and known environment.

ロボットのプログラミングでは、リアルタイムでの結果出力が非常に重要になってきている。6 d.o.f.マニピュレータでは、障害物回避経路の計画は時間がかかる作業なので、通常シミュレーションで処理される。ここでは、6個の外巻ジョイントを持ったマニピュレータでグロス・モーション・プランナーを実現させるために採用した、幾何学モデルと推論技術について説明する。最初に、ロボットの作業空間の表現方法に起因する問題の解釈について説明する。次に、我々が考案したC-space表現を使って実際に行った、軌道解析について説明する。この場合、全てのC-spaceを計算するのではなく、最初の2つのリンクに対してのみC-spaceを定義する逐次法を採用している。その位置がわかっており静止している障害物の近似により、計算時間が大幅に短縮されるので、プランナーの幾何学的推論作業は大幅に軽減される。今回の研究は、理論研究やシミュレーションだけに留まらず、PUMAロボットを使った様々な実験により、開発したソフトウェアの実際の効果と実用度を評価する。実験は、静止している障害物の位置が予め判っている場合についておこなっている。

*To whom all correspondence should be addressed.
†Present affiliation: BTicino S.p.A., Varese, Italy

Journal of Robotic Systems 12(2), 93–104 (1995).

CCC 0741-2223/95/020093-012

1. INTRODUCTION

In the wide area of robotics, a relevant research effort is the development of some intelligent capabilities of robots, such as sensing and autonomous motion planning. The ultimate goal is to provide the robot with such control devices that it can accomplish a variety of complex tasks, under conditions incompletely known, assigned in terms of desired goals, with little specification of the actions the robot must perform. This is really the opposite of what is the practice in robot programming, where one must describe every single movement.

Automatic robot programming requires equipping the robot with autonomous sensing and decision-making devices. This is advanced robotics (as opposed to industrial robotics), which deals with robots interacting with the outside world, and therefore requires artificial intelligence (AI) techniques.

A class of problems whose solution is essential to increase robot autonomy is motion planning.[1] For a manipulator, this problem actually consists of finding a sequence of elementary movements, capable of leading the end-effector to the desired points while avoiding any possible collision with the obstacles in the working area. In pick-and-place operations, motion planning takes different aspects for gross motion, i.e., searching for collision-free paths, and fine-motion, i.e., local searching of safe object grasping movements (which usually involves sensory feedback devices[2]).

Motion planning is the subject of many articles,[3,4] but, unfortunately, its complexity has limited the real applications of these theoretical studies. The widest statement of the problem, known as *generalised movers' problem* or *find-path problem*,[5] consists of planning a collision-free path for any robot with any number k of degrees of freedom (dof), and also deals with interactions among many robots. Schwartz and Sharir[3] demonstrated that this problem can be solved in polynomial time with the number n of algebraic constraints defining the set of allowed configuration of the robot, but it is doubly exponential in k.

Canny[6] proposed an algorithm ending in single exponential time with k, using direct techniques easily adaptable to various computational geometry problems, especially in multidimensional cases.

To be useful, a planning algorithm must run in a reasonable amount of time. So researchers suggested heuristics and search strategies to overcome complexity.

In fact, many algorithms are based on C-space, the k-dimensional space defined by the variables related to the dof of the robot, which allows an explicit characterization of the constraints imposed on the motion. This involves two big problems: (1) high number of dimensions of the C-space, and (2) complex representation of the obstacles in this method.

Reducing complexity had a great role in the work of Tomas Lozano-Pérez, with appreciable results. He began studying planar problems with two[7,8] and three dof,[9] then arriving at a more general system.[2] Similar is the approach of C. Laugier,[10] and of Xing,[11] who used an oct-tree representation of the C-space. Gupta[12] tried to defeat complexity by a sequential approach, reducing an n-dimensional problem to one mono-dimensional plus $(n - 1)$ bi-dimensional problems.

Our algorithm[13] also uses the C-space approach. Motion planning articulates at two main phases. The first one is about the construction of a problem-oriented representation of the robot working space. The second one is about the actual trajectory calculation, carried out in that representation. A good modeling of the environment is crucial for the efficiency of the subsequent movement determination. As we aimed at developing a system really usable in practice, we didn't try to calculate the whole C-space representation. The strategy is sequential, starting from the first link of the robot and adding a link at a time. We actually determine the C-space for the first two links while, for the other links, we find acceptable intervals considering the path already determined by the calculations of previous links. This type of strategy was already experimented with by Lozano-Pérez,[14] but with the whole calculation of the C-space and the consequent inefficient management of the same. Similarly, Gupta[12] adopted a discretization of the angular intervals of a link, with consequent increased computations, but we use this solution only when the demands of precision are high.

A second trick, which greatly speeds the computation, consists of the particular approximation of obstacles (transformed into tori) that allows us to reduce the problem to planar geometric reasoning. This method was recently adopted by Lozano-Pérez,[2] but not by Gupta.

As we consider articulated robots, like the Unimation PUMA, we take advantage of the configuration of the first two joint axes, which are perpendicular, and approximate the obstacles with tori centered at the origin of the robotics system. This allows us to consider the arm as planar and to oper-

ate in a plane (determined by the value of the first link), obstructed by obstacles corresponding to cross-sections of tori.

The determination of movements was reduced to a graph search problem. The nodes of this graph represent a particular configuration of the robot and we can connect two nodes if and only if the robot can move from one configuration to the other without colliding with the obstacles. More insight about the method implemented to find sequences of optimal joints values will be given in the following.

We have performed experimental tests on the Unimation PUMA 500 at the Robotics Lab of Politecnico di Milano. The strategy was quick and powerful in most realistic applications, and is apt to be the basis for a whole planning module.

This article is organized in four sections. In sections 2 and 3 are presented the concepts on which our theory is based. The first part deals with the modeling methods; the second explains the technique and the heuristics of path searching. In the section 4 are described the implementation status and an experiment that will show most capabilities of the system. We end with some considerations and suggestions for further developments.

2. MODELING THE WORK SPACE: GEOMETRIC MODELS AND C-SPACE

We have created the geometric models necessary for motion planning after considering the required amount of computation for geometric and topological properties. The obstacles are defined by sweeping polygonal bases along straight lines to generate right prisms. We can obtain more complex solids matching more right prisms. For example, a cube can be defined by specifying:

- the polygon that represents its horizontal section base ((0.0)(0.1)(1.1)(1.0))
- two values indicating the z-coordinates of the vertical sweeping line 0, 1

To geometrically describe the manipulator, the C-space approach[8] allows us to obtain a representation of every configuration of the robot system in terms of a vector in a k-dimensional space (if k equals the number of degrees of freedom of the system). So, considering a robotic system A, we call C_A the set of possible values of the configuration vector c.

We are especially interested in all the safe configurations of the robot, i.e., the configurations that don't cause collisions between the robot itself and the obstacles. In accordance with many authors, this space is termed free-space FS_A. Finally, we call $CO_A(B_i)$ the projection of the obstacle B_i in C_A. The free space characterized:

$$CO_A(B_i) = \{c \in C_A: A(c) \cap Bi \neq \phi\}$$

$$FS_A = C_A - U_{i=1,m}\, CO_A(B_i)$$

where c is a generic configuration of A, and m is the total number of obstacles present in the work area.

FS_A is represented by a non-oriented graph GM_A (N, A), whose nodes are portions of this space, specifying the angular ranges for every parameter p_i of the configuration vector c. If $n_i \in N$ and $n_j \in N$, we draw the arcs a_{ij} and a_{ji} if and only if the robot can move from $c_i \in n_i$ to $c_j \in n_j$ (and vice versa). As both nodes belong to FS_A, the movement is accomplished without colliding with the obstacles. The start position of the robot corresponds to a node of the graph. If the goal belongs to the graph, the system heuristically finds a *quasi-optimal* path that leads the robot from the initial configuration to the final one.

2.1. Transforming the Obstacles

The system starts from the description of the working area in terms of obstacles defined as prisms. If obstacles are few and sparse, we may loose portions of the free space without problems. Here, a rough representation of the obstacle as a prism that is a geometrical hull of the real object is adequate. Otherwise, the obstacles can be divided into more prisms, and the difference between the real object and the resulting prisms will be reduced, obtaining a more realistic representation, obviously at the price of augmenting the number of prisms and thus the computation time.

The prisms may be eventually increased to account for measure errors.

2.2. Constructing the Slices

The planner approximates the obstacles by tori, with constant cross-section, centred in the origin, as illustrated in Figure 1. This allows the work-area to be divided into sectors that are angular intervals. Sectors in which the sections of the tori are constant are sectors called *slices*. Their construction occurs in two phases: (1) sector determination and (2) deter-

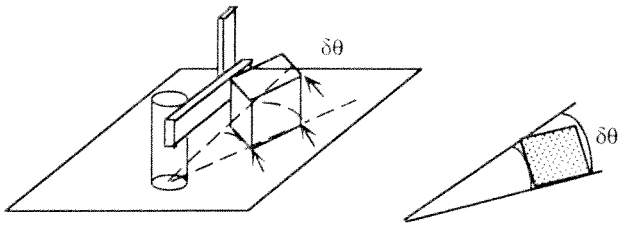

Figure 1. Transforming the obstacle into a torus with constant cross-section (arrows indicate width and height of the section).

mination, starting from sectors, of slices themselves.

We generate slices by rotating the arm "righty" (for a PUMA robot this indicates a range of values for θ_2) and in its maximal extension in θ_1 plane, and by considering its collisions with the polyhedral obstacles.

Notice that, for a given value of θ_1, the colliding obstacles are also in positions diametrically opposed to the rotation center when the robot is "lefty," as seen in Figure 2.

A sector represents an angular range of θ_1, associated with two lists of prisms. The first, called *main prisms*, contains the obstacles colliding when θ_1 assumes values in the range and the robot is "righty" and the second one, called *secondary prisms*, those obtained when the robot is "lefty".

2.3. Sectors Construction

We start determining, for every base of every prism, its *angular obstruction* $CO_A^1(B_i)$, i.e., the angular range of θ_1 for which the arm penetrates the base of the obstacle B_i, where B_i is the considered prism.

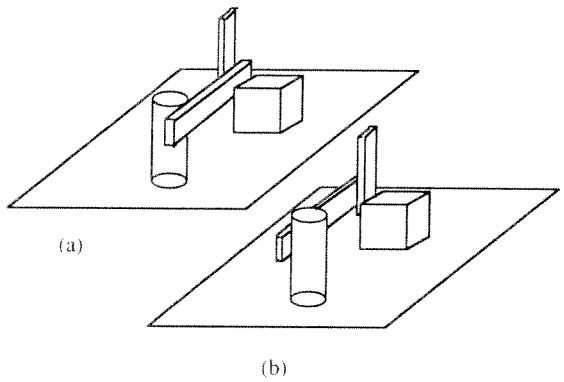

Figure 2. (a) Collision with the robot in righty configuration; (b) in lefty configuration.

The arm is described as a list of consecutive vertices corresponding to its projection on the plane of θ_1 when it is in maximal extension on this plane (this is a 2D problem). Considered separately are the part of the arm going from the rotation center to the end-effector, and the one in the opposite direction (*tail*).

2.4. Invalid Intervals for a Joint

We need to determine the angular obstruction of an obstacle for a joint, i.e., the projection on the considered joint variable q of the image of B in C-space, called $CO_A^q(B)$. We want to determine the range of values of q causing a collision between the mobile system A and the obstacle B,

$$q\colon A(q) \cap B \neq \phi$$

To do this, we must know the geometric description of the link connected to the joint, generically called *rotating object*, as we consider links of the robot and transported objects.

The system uses a description of the arm obtained from projecting, in the planes perpendicular to the rotations axes of the first three joints, a polyhedric body over-approximating the arm. We call A that polyhedron, and A^q its projection in the rotation plane of the joint q. For simplicity, these descriptions are given as the vertices of consecutive vertical or horizontal lines, as illustrated in Figure 3.

A^q is subdivided into two parts, called *front description* and *back description*, as shown in Figure 4.

This subdivision is made considering as positive the anti-clockwise rotations and dividing A^q into the two parts that arrive first or later during rotation. Of course, as for the obstacle, we consider the projection in the plane of q, B^q, as a polygon.

There can be have two kinds of collisions between the obstacle and the rotating object: those

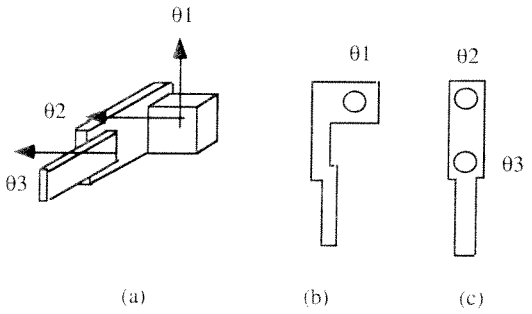

Figure 3. (a) The first 3 links are approximated; (b) projection of the three links on the plan orthogonal to $\theta 1$; (c) projection on the plane orthogonal to $\theta 2$ and $\theta 3$.

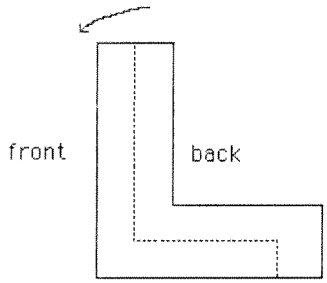

Figure 4. Front and back descriptions for the rotating object.

caused by a vertex of the rotating object colliding with an edge of the obstacle, and those caused by the contact of a vertex of the obstacle and an edge of the rotating object. The cases of contact edge to edge or vertex to vertex are assimilated to the first two. The *1st-type contacts* are those between A^q vertices and B^q edges, and *2nd-type contacts* the others, as illustrated in Figure 5.

First to be calculated are the *reference angles*, defined as the angles defined by the lines connecting the contact points to the rotation center and the horizontal line. They are called 1st- or 2nd-type angles according to the points that generated them. The reference angles of the 1st-type angles are computed determining the intersections between the B^q edges and a rotating segment as long as the rotating object. The 2nd-type angles correspond to the angular values of B^q vertices in polar coordinates.

These angles are corrected to consider the dimensions of A^q in the following way (see Fig. 6):

- by calculating the reference angles α_r and the distances d_r from the rotation center of the respective contact points (i.e., polar coordinates of the points);
- by calculating the *correction angles* α_f and α_b obtained considering the angle from the reference, and the radius d_r at the intersection points belonging to A^q_f and A^q_b, respectively;

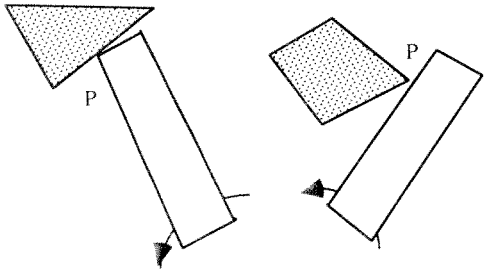

Figure 5. First and second type collisions.

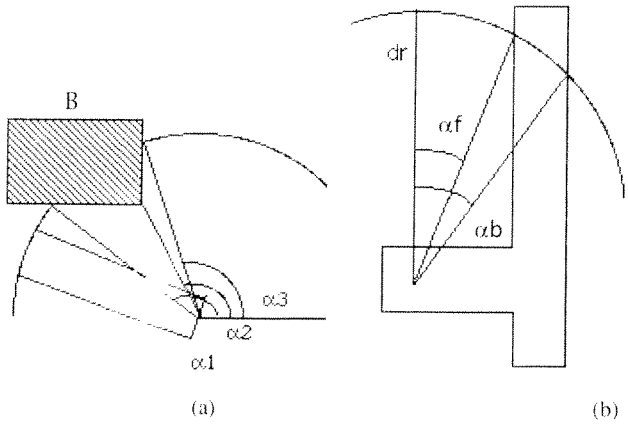

Figure 6. (a) The reference angles for the obstacle B; (b) the front and back correction angles for the rotating object, at the distance dr.

- by computing the actual angles by algebraically adding α_r, α_f, and α_b.

From the actual angles we choose the two that contain all the others, and these become the extremes of the interval for q defining $CO^q_A(B)$ (see Fig. 7).

2.5. Main and Secondary Prisms

As already stated, we must consider both right and left configurations of the robot, which generate two lists of angular obstruction (Fig. 8) starting from one list of obstacles.

Generally, these intervals are fully or partially laid one upon the other, because for one value of θ_1

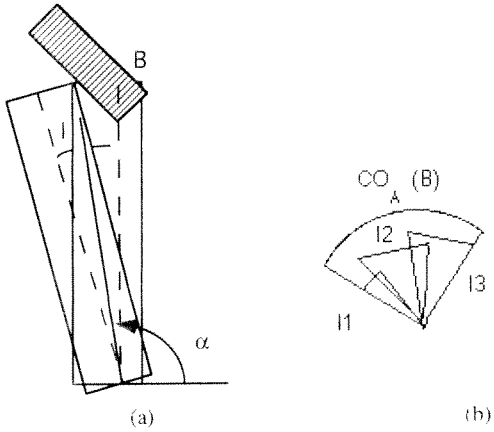

Figure 7. (a) Every reference angle modified with the correction angles generates an interval I; (b) the interval containing all the intervals computed for an obstacle is $CO_A(B)$.

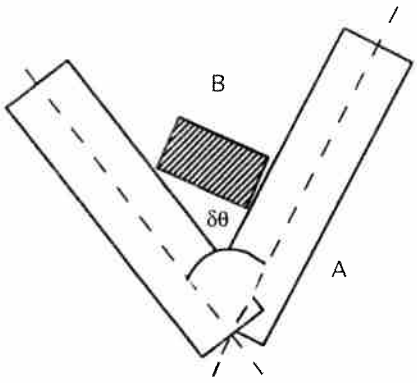

Figure 8. The angular obstruction for the obstacle B is $\delta\theta$.

the arm may interfere with more than one prism. So we eliminate these superimpositions, creating new, smaller, consecutive intervals corresponding to all the parts, laid or not, of the starting intervals, as illustrated in Figure 9.

These new intervals are related to a list of prisms, divided into main prisms and secondary prisms according to the arm configuration.

2.6. From Sectors to Slices (Tori Determination)

After computing sectors we compute tori. The obstacle section as seen by the rotating object (i.e., the projection of the obstacle on the θ_1 plane containing the rotating object) varies with θ_1. The cross-section of the tori will be calculated as the minimum rectangle containing these real sections. For a given sector and a prism belonging to it, the corresponding torus is given by the torus, with a rectangular cross-section that extends for $\Delta\theta_1$, the sector interval (see Fig. 10).

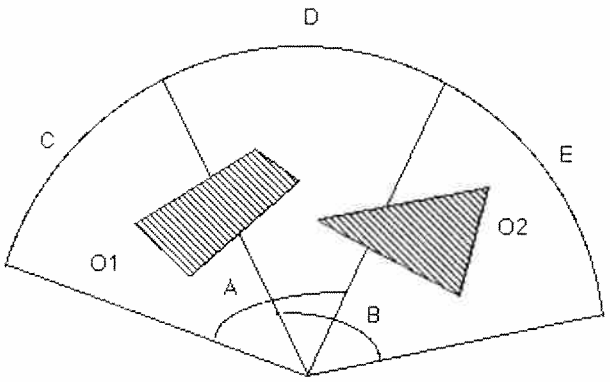

Figure 9. Subdividing the intervals A and B into C, D, and E.

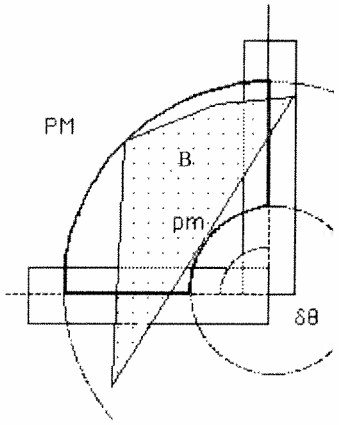

Figure 10. The torus approximating the obstacle B in the interval $\delta\theta$.

The real problem consists of calculating this rectangular section. Its height is the same as that of the prism generating the torus. Its width must represent all the possible interferences between the arm and the prism in the considered interval. It corresponds to the minimum and maximum distances of all the contact points between the prism and arm from the 2nd-link rotation axis.

In fact, we are considering the projections of θ_1 plane of the arm and the prisms, so we can reformulate the problem as the research, in the set P_c, made of all the contact points between a fixed polygon (the base of the obstacle) and a rotating one (the arm), of the points p_m and p_M of minimum and maximum distance from a given axis (2nd joint rotation axis, i.e., the straight line normal to the arm). The set P_c is infinite, but we can calculate a small and finite set of meaningful points, surely including p_m and p_M; they are:

- the intersections between the base and the arm, when it is in the initial position θ_i of the interval;
- the intersections between the base and the arm when it is in the final position θ_f;
- the vertices of the base laying between the arm in the initial and in the final position;
- the intersections between the perpendiculars from the center to the base edge laying between the arm in the initial and the final position;
- the rotation center, if the base contains it.

Also to be considered is that the straight line from which to calculate the distances rotates with θ_1 in the interval $\Delta\theta_1$ (a bundle of straight lines). The

research can be limited to the two extreme lines of the bundle.

The research of meaningful points is speeded up if the geometric description of the arm is divided, as in our scheme, into a front and a back part. The points making up these parts are, more or less, half of those of the global one. The meaningful vertices and points lay between the back description in the position corresponding to θ_i, and the fore one at θ_f. Also, the meaningful intersections lay between the front description in the position θ_i and the back one at θ_f. This trick allows computation savings.

When we have all the meaningful points, we must calculate their maximum and minimum distances. If the base extends beyond the working area of the arm, we choose as maximum distance the maximum arm length. So, the torus is completely determined.

3. COMPUTING THE PATH FOR THE FIRST THREE JOINTS

Let's consider, how it is possible to get the sequences of robot joints values starting from the particular world modeling we carried out.

3.1. Graph Nodes Determination

Having divided the world into slices according to the obstacles for the first joint, the problem becomes how to move the second and third links into these slices (we will discuss end-effector orientation only later).

This is a planar-type problem. In fact, having modeled constant cross-section tori, we may consider the arm as a two-link planar arm moving in a plane. The obstacles in this plane depend upon θ_1 values, i.e., upon the slice, and are represented by the (constant) section of the tori in that slice. Resuming, in the plane are:

- a rotating polygon that is the projection on the θ_1 plane of the second link, L_2;
- a second rotating polygon, L_3, corresponding to the projection, on the same plane, of the third link, and tied to the end of L_2;
- some polygons, representing obstacles, (tori cross-section).

The rotation angles of the two links are θ_2 and θ_3. We must now determine legal values for θ_2 and θ_3, i.e., FS_A^{23}. This is carried out only for θ_2; the computation of θ_3 will be performed only when strictly necessary.

3.2. FS_A^2 Calculus

Generally, legal (i.e., collision free) values for θ_2 depend also upon θ_3. Calculation of the θ_2 intervals where L_2 penetrates into some obstacle is illustrated in Figure 11. The set of these intervals is the *forbidden zone*; the remainder is the *free zone*. The procedure is the same used for θ_1 and already described.

For some free intervals for θ_2, there are possible collisions between the third link and the obstacle. Therefore, we have to build other intervals, α, to account for that. The value for α is easily determined if the second link is longer than the third, as in the case of the PUMA robot. These new intervals are called *mixed*. Any interval that is neither forbidden nor mixed is called free. In the free intervals θ_2 can assume any value independently from θ_3. Usually the intervals are partially superimposed; we combine them to get smaller intervals with the most restricted definition (see Fig. 11).

The next step is the construction of the graph nodes. A node is characterised by three angular intervals, respectively related to θ_1, θ_2, and θ_3. The interval related to θ_1 identifies the slice; the interval for θ_2 comes out from the zones subdivision just described; θ_3 is not specified, at the moment, unless the corresponding interval is free.

The nodes whose corresponding intervals are superimposed or adjacent are linkable. These nodes constitute an efficient representation of the free space, and, linked in the graph, allow quick searching of the free path from start to goal.

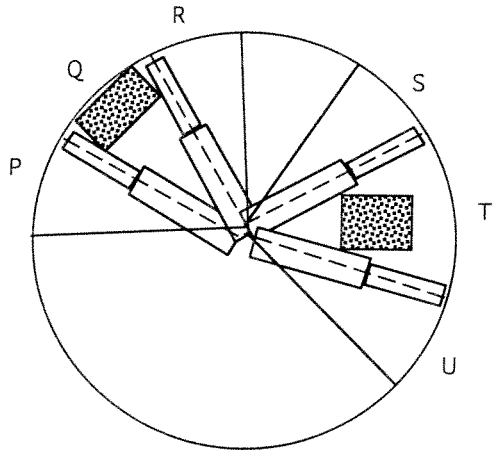

Figure 11. *T* is a forbidden interval for the second link; *Q* is a mixed interval; also *P*, *R*, *S*, *U* are mixed intervals.

3.3. Building the Path for the First Three Links

Path search is performed in a different way if the robot moves alone than if objects are transported. In the first case all graph nodes are built once; in the second case it is necessary to recalculate nodes whenever objects change their position (because the geometric models used to calculate slices and tori change). So, it's useful to expand the graph during path search.

Moreover, the path on the graph doesn't correspond to the final result of all calculations. The path on the graph only corresponds to a sequence of the consecutive intervals where the values of the actual link variables belong. The final result will be a sequence of points in the C-space that are the vertices of a polygonal line corresponding to a series of robot safe configurations. This sequence of points is the *actual path*, not to be confused with the *graph path*.

The path is computed in two phases: (1) computation of the optimum graph path; (2) computation of the actual path.

3.4. Searching a Path in the Graph

To compute an optimum path, it is necessary to specify the optimality criterion, so it's necessary to determine the crossing cost of every node. To minimize the time the arm spends to accomplish the movement, we get the optimum path minimizing the total sum of variations carried out by each joint. In practice, this is to minimize the distance between start and goal in the free space. The time spent to move from one configuration to another corresponds to the maximum travel time spent by any joint to move (at the maximum speed) between the joint values of two configurations, and it is directly proportional to the difference between the two same values (not considering the discontinuities from 2π to 0).

A simple method to calculate a node crossing cost is to sum all the amplitudes of the node intervals. This simple method is not really usable; in fact, crossing a node doesn't imply that joint variables assume all the values of the related interval. Generally, the opposite is true, because minimum search requires that we "stay" in a node as brief a time as possible before going to the next one.

A cleverer analysis shows that to be able to move from a node to the following node, the joint variable must assume the values belonging to the superimposed interval of the relative nodes intervals. A method consists in calculating the cost as the distance from the presently reached point to the mean point of the superimposition between the present node and the next one. This will be taken as the base point for the following computations. This method is actually used in our system, and the smaller the intervals, the more accurate it is.

In searching for the optimum path the A* algorithm was used.[15] The estimation, that represents heuristic information, is taken as the minimum distance between the mean point and the goal.

3.5. From Optimum Path to Actual Path

The actual path calculation is carried out first for θ_1 and θ_2 and then, using the obtained results, for θ_3. The problem to be solved is: given n intervals I_k, at least superimposed, and two points, start and goal, find the sequence of points p_k ($p_k \in I_k$, $k = 1 \ldots n$) that optimally arranges the path from start to goal (see Fig. 12).

The solving algorithm path (start, goal) has the following steps:

- find the ideal path P_i from start to goal, i.e., the path that equally divides variations between the points
- if all the points in P_i belong to the relative intervals, then return P_i
- or else find the point q that is farthest from its interval and apply path (start, q), and path (q, goal).

Once θ_1 and θ_2 paths have been calculated, we can think of θ_3. We know the values of the first two links, the intervals of the other variations, and the rotation center of the third link. Each interval for θ_1 denotes a slice, so some tori corresponding to the obstacles. There is still a problem because θ_2 isn't fixed, but it varies inside its interval. This implies

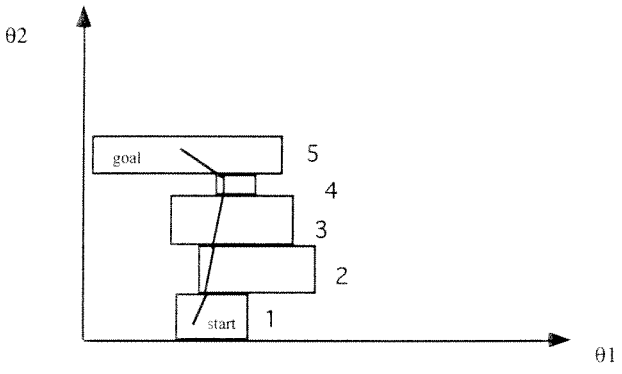

Figure 12. Graph path and actual path.

that the rotation center of the third link is also variable.

This difficulty can be overcome by establishing that θ_2 assumes the mean interval value and propagating its variation on the obstacles. Specifically, one must rotate every obstacle vertex by $(\Delta\theta_2)/2$ both in the clockwise direction and counterclockwise, and consider the two extreme points obtained by these rotations. The envelope polygon formed by all these points will be taken as the obstacle for computing the interval for θ_3, as illustrated in Figure 13. This is an excess approximation.

To compute free intervals for θ_3, we again use the described method is again used. Sometimes we generate intervals for θ_3 neither superimposed, nor adjacent. This is because the arm can't actually pass through, or because the propagation of the θ_2 variation has excessively widened the obstacle. We can successively try to divide $\Delta\theta_2$ into two parts. Doing so, the propagated obstacles lessen and $\Delta\theta_3$ widens. After building the sequence of the θ_3 intervals, we can search the actual points as we already did for θ_1 and θ_2.

3.6. End-Effector Orientation

Having determined the values of the first three joints, we have fixed the end-effector position in the space. We must, next, establish the orientation of the reference frame fixed in the hand with respect to the world reference system.

To specify orientation we adopt the Euler angles. Given a reference coordinate frame, any orientation is described in terms of a rotation by an angle ϕ around the z axis, followed by a rotation θ around the new y axis, and finally by a rotation ψ around the new z axis.

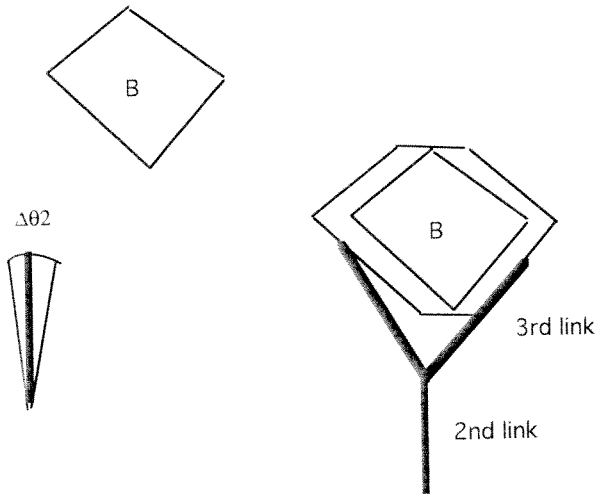

Figure 13. Propagating the variation interval of $\theta 2$ to grow the obstacle B for the third link.

The problem of finding end-effector orientation is so formulated: given initial and final values for ϕ, θ, and ψ, together with the already available sequence of values for the first three links, find the sequence of values of ϕ, θ, ψ.

The solution consists in applying the same method used for the first three joints; in fact, the rotation axes of ϕ and θ are positioned, in respect to one another, as θ_1 and θ_2 axes. The only difference is that now the reference frame is not fixed to the robot base, but roto-translates with the robot hand as θ_1, θ_2, and θ_3 change.

3.6.1. Path Calculation for the First Two Euler Angles

Consider a reference frame H attached to the third robot link and oriented as the hand frame (according to the usual conventions) when the Euler angles are null. The computation of ϕ and θ are the same as for the first two joints. The only difference is that the rotating object is different.

After computing the sequence of values for θ_1, θ_2, and θ_3 we examine couples of consecutive terms of values. We calculate the mean values of these intervals and then the direct cinematic transformation. The points to be reached by the end-effector are obtained, and these points are taken as the consecutive origins for the frame H.

It is necessary to remember that we consider mean values, and then we must propagate the actual values variations. It is the same problem discussed about θ_2 variation, but now it's a three-dimensional problem. After the frame is roto-translated, we have a description of the world as "seen" by the hand for particular values of θ_1, θ_2, and θ_3. To avoid useless efforts, we remove all obstacles that are out of the end-effector reachable area before making any calculation (rarely are all obstacles removed).

Now we are ready to apply the usual algorithms and to calculate the tori, graph, etc. There are groups of tori and groups of nodes. To build only one graph, we choose the consecutive nodes corresponding to the path already found for θ_1, θ_2, and θ_3. This can be achieved simply by building an oriented graph whose arcs go from preceding to following groups of nodes, without linking nodes belonging to the same group. On this graph the

optimum path search is performed using the A* algorithm.

3.6.2. Path Calculation for the Third Euler Angle

The general idea followed is always the same. This time, we consider a reference coordinate frame H' so that last link rotates around the z axis: it corresponds to frame H rotated by ϕ around the z axis and then by θ around the new y axis. As usual, we must propagate ϕ and θ variations; this agrees with what we have done for θ_1 and θ_2 in the frame H'. Now we can calculate the free zone, the following intervals and, finally, the actual path (as for θ_3). The trajectory computation is finished.

4. IMPLEMENTATION AND EXAMPLE

The planning algorithm has been implemented and experimented with at the Robotics Lab of Politecnico di Milano. The program was first written in Golden Common LISP 3.0 on a IBM PS/2 and tested on a Unimation PUMA 500. Implementation details are found in Massa and Negretti.[13] Following is a description of the phases of the path construction problem.

1. *Initialization:* The robot and the obstacles data structures are defined in memory, taking data from a file that describes the robot geometry and parameters and a file containing the obstacles. The needed projections are generated.
2. *Nodes construction:* Slices are constructed and free areas are determined. Therefore, a graph structure containing the nodes and their relations is built.
3. *Optimum path search:* A* is applied to the graph.
4. *Actual path search for the first three joints:* From the superimposed intervals for the first two joints the free intervals are computed for the third joint. The procedure is iterative; if an interval is not found for θ_3, the intervals for the second joint are more and more subdivided. If a solution is not found, we go back to (3) to search a new path.
5. *Path search for the last three joints:* Computed the rotation centers for the wrist, the propagations, and the slices, then the optimum path for the last three joints, and the actual path for ϕ and θ. For the last angle Ψ, we must propagate the variations, roto-translate them for the mean values, and then compute the free intervals.

The path obtained can be used in a VAL program to move the PUMA arm, or tested in simulation using SIMULATE,[16] a robot simulator developed on the geometric modeller WM.[17]

In the series of figures shown in Figure 14 (obtained using SIMULATE) there is a representation of a manipulation task performed by the robot and planned by our algorithm. The robot avoids the column between the start and the goal and, instead of rotating only the first joint (as it would do without obstacles), it carries out a wide first and second joint rotation, going from a right to a left configuration. One can also appreciate the fact that the end-effector goes very near to the obstacles to reduce the cycle time without problems. This planning required about 1 hour and 45 minutes of computing time. A significant improvement (cutting the time in half) has been obtained, at little cost, using the Allegro Common Lisp on a 486-based PC. In this environment we have carried out many other successful experiments. The same program was also used to compute trajectories of another articulated robot, the CRS 460. For this robot we had to remove the difference between primary and secondary prisms.

The final aim of the system was to provide a trajectory planner for a maintenance robot. The final test-bed will be a mobile robot carrying a manipulator and moving in a power plant to check valves and to open or close them after reading pressure or temperature values on on-site instruments. For this application we need to cross areas with pipes and to manipulate valves, moving the arm in an area moderately cluttered with pipes and instruments.

5. CONCLUSIONS

The aim of this study was to solve a gross motion problem for an anthropomorphic manipulator with six dof, put in a fixed and known environment. The basic knowledge is composed of work area and robot descriptions; the algorithm produces elementary geometric movements for the joints of the robot to move the end-effector to the desired position and with the desired orientation, avoiding any possible collision with both the robot and the transported object.

We wanted to obtain a system that could really

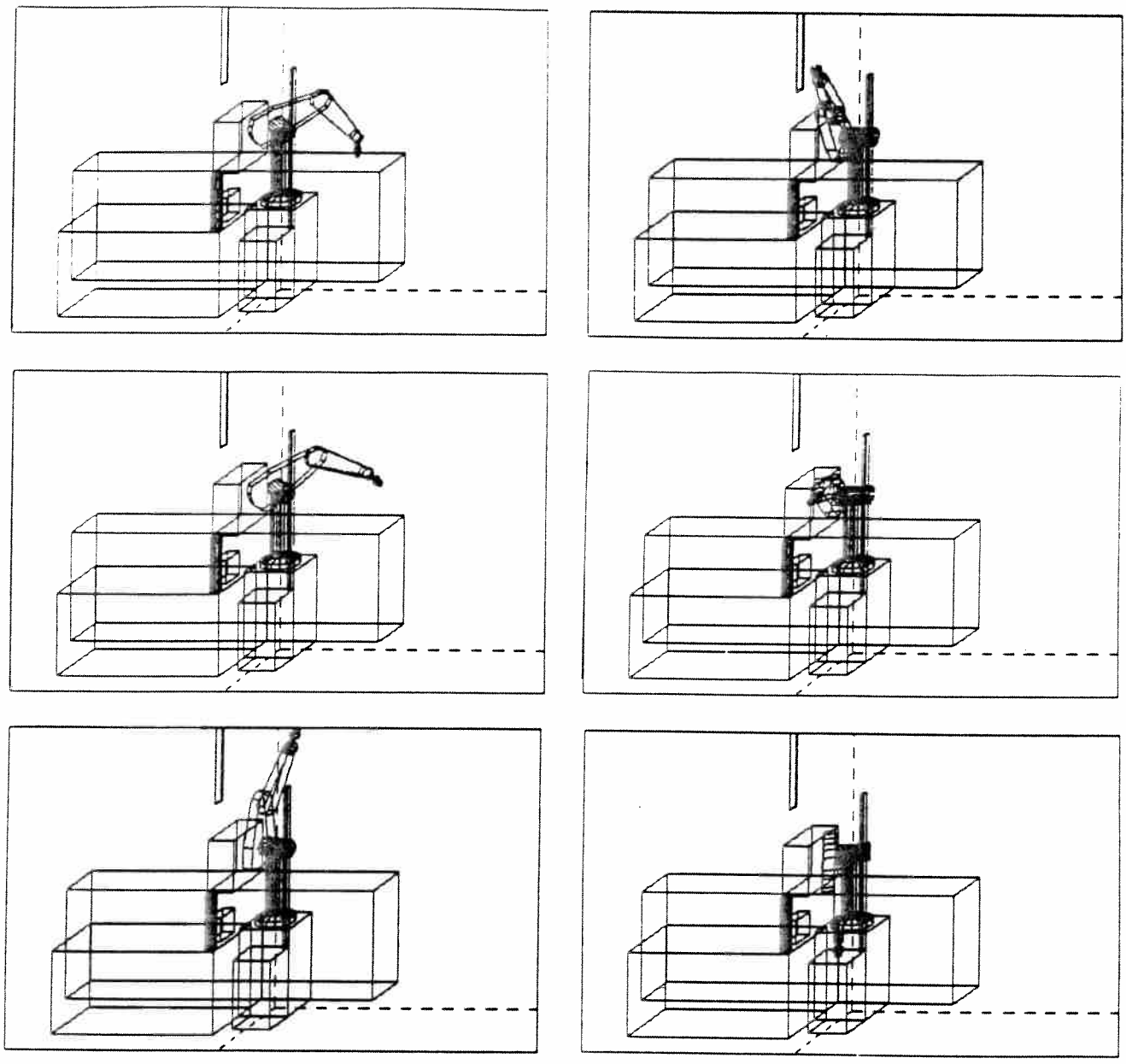

Figure 14. Manipulation task performed by the robot.

be used, and give good results in a reasonably short time. This goal contrasts the real nature of the problem, defined NP-hard, as indicated before. Therefore, we have adopted some approximations and tricks, balancing efficiency and precision, and trying, however, to preserve the system capability to reach the goal.

Specifically, we adopted a powerful method in knowledge representation, consisting in approximating obstacles with tori. This fact allowed us to solve three-dimensional collision-detection problems reasoning on simpler 2D problems, with a great decrease in complexity. The technique of *divide et impera* reduced the dimensions of C-space (which is, in the six dof problem, 6-dimensional). Thanks to a sequential approach, consisting in calculating paths hierarchically, starting from the first degree of freedom and ending with the last one, the whole C-space computation could be omitted. Instead, we consider a joint after the values of the previous ones are determined, and so in limited and well-defined regions. The internal representation of C-space,

based on graphs, permits fast search algorithms, too. So we could reduce our computational effort and still be able to solve most of the proposed problems, in situations of quite high complexity and in reasonable times (one or very few minutes for complex manipulations, about 10 seconds less if no manipulations are requested). Yet there are situations in which the system fails to find a solution. These situations are usually characterized by the presence of many obstacles of large dimensions. In this case, it may happen that no consecutive free intervals are found and no planning is possible. We could reduce the angles in computing the slices, but sometimes this makes the system absolutely too slow.

The idea of transforming obstacles into tori was suggested by Lozano-Pérez,[2] but with the complete construction of C-space. The techniques used in the transformation are original and are always aimed at reaching efficiency, as the original trick of considering half force and half back descriptions, or the classification of full, free, and mixed zones. Lozano-Pérez[18] has presented a parallel method for computing configuration space maps.

A sequential strategy has been introduced by Gupta,[12] but he calculates the free space for a joint with a subdivision of the intervals of the previous joints, and his results are obtained in simulation.

We want to underline that our work has a place among the few actually operating ones; it is not limited to theoretical studies or simulations. The experimental check of its operation has been run thoroughly, with various and numerous tests, made easy by the real efficiency and suitability of our system. In these checks we have reproduced realistic situations, and we estimate that in about 10 minutes one can create a description of the whole robotics environment suitable for the system. The final movement results are fluid and come up to our expectations.

It seems easy to broaden the system to consider redundant robots as well, because the squential approach of the strategy permits us to reuse the already developed algorithms.

Partial support for this research has been obtained through the National Project on Robotics of the Italian Research Council (PFR of CNR), under the objective "ALPI."

REFERENCES

1. J. E. Hopcroft, J. T. Schwartz, M. Sharir (Eds.), *Planning, Geometry, and Complexity of Robot Motion*, Norwood, NJ: Ablex Publ., 1987.
2. T. Lozano-Pérez, J. L. Jones, E. Mazer, P. A. O'Donnell, "Task-level planning of pick and place robot motions," *Computer*, March, 21–29, 1989.
3. J. T. Schwartz, M. Sharir, "A survey of motion planning and related geometric algorithms," *Artif. Intell.*, **37**, 157–169, 1988.
4. M. Sharir, "Algorithmic motion planning in robotics," *Computer*, **22**(3) 9–20, 1989.
5. J. H. Reif, "Complexity of the movers' problem and generalizations," Proc. 20th IEEE Symposium on Foundations of Computer Science, 1979.
6. J. F. Canny, "The complexity of robot motion-planning," Cambridge, MA: MIT Press, 1987.
7. T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst. Man Cybern.*, **SMC-11**(10), 681–698, 1981.
8. T. Lozano-Pérez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.* **C-32**(2), 108–120, 1983.
9. R. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for findpath with rotation," *Proc. IJCAI 83*, San Mateo, CA: Morgan Kaufmann, pp. 799–806.
10. C. Laugier, "Planning robot motion in the sharp system," in *CAD-Based Programming for sensory Robots*, B. Ravani (ed.), Berlin, Heidelberg; Springer-Verlag, 1988.
11. D. M. Xing, "Collision-Free Trajectory Generation For A General Robot Manipulator", Tech. Rep. CRIIF, Laboratoire de Robotique, Paris, France, 1987.
12. K. K. Gupta, "Fast collision avoidance for manipulator arms: A sequential search strategy," *IEEE Trans. Rob. Autom.*, **6**(5), 1990.
13. R. Massa, R. Negretti, "Pianificazione efficiente di traiettorie prive di collisioni per robot manipolatori: Studio e sperimentazione," Master Thesis in Electrical Engineering, Politecnico di Milano, 1991.
14. T. Lozano-Pérez, "A simple motion planning algorithm for general robot manipulators," *IEEE J. Rob. Autom.* **3**(3), 1987.
15. P. E. Hart, N. J. Nilsson, B. Raphael, "A formal basis for the heuristic determination on minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, **SSC-4**, (2) 1968.
16. G. Gini, "Simulate: A robot simulator," Internal report, Dipartimento di Elettronica, Politecnico di Milano, 1992.
17. C. Mirolo, E. Pagello, "A solid modeling system for robot action planning," *IEEE Comput. Graphics Appl.*, **9**(1), 55–69, 1989.
18. T. Lozano-Pérez, "Parallel robot motion planning," *Proc. IEEE Int. Conf. Rob. Autom.*, Sacramento, CA, April 1991.