Modelling Robot Manipulators in a CAD Environment Using B-Splines

Colin G. Johnson and Duncan Marsh.
Department of Mathematics,
Napier University, 219 Colinton Road,
Edinburgh, EH14 1DJ, Scotland, UK
Telephone : +44 131 455 4631
Fax : +44 131 455 4232
email : colinj@maths.napier.ac.uk, dmarsh@maths.napier.ac.uk

Abstract

A major aim of robotics research is the provision of systems which simplify the programming of robots, enabling experienced designers and engineers to implement robotic devices as part of a larger systems without the need to become expert programmers. Also in the quest for a flexible industrial production system is it desirable to be able to reprogram robots offline, so that they can be doing one task whilst being prepared for another. This paper describes the mathematical and computational background to a system which enables the development and testing of robot programs in a CAD environment in a way that is simple to use and compatible with existing methods used in CAD systems.

1. Introduction.

One of the major themes of robotics research since its inception has been the development of intuitive methods of programming robots which empower experienced designers and engineers to implement robotic systems without the need to become expert robot programmers. In recent years the idea of programming robots in a "what you see is what you get" style environment has become an important strand of development, using CAD models of the robot and its environment to enable the motions of the robot to be tested in a simulated environment before it is used in practice [8, 6].

However current systems use object representations which are good for representing static objects, yet which can only represent moving objects approximately, or by using a different representation from the static objects. What is needed is a system which can make use of a single representation for all the objects found within it, including the robots themselves, the other objects within the robots' environment, the motions of the robots and the regions swept out as the robots and mechanisms follow those motions.

Using this single representation it should be possible to combine these various objects using efficient algorithms for functional composition, projections, extrusions, tensor-product structures and other ideas from geometry and computer-aided design. These few algorithms could then be implemented very efficiently and rigorously tested, rather than many "special case" algorithms having to be written as is often the case in robotics (for example the various kinds of contact described in the appendix to [26]). Indeed this small set of basic algorithms could be implemented in hardware, as has proven successful in the development of computer-graphics workstations.

The power of such a unified system has been advocated strongly in the solid-modelling community, for example this paragraph by Farouki and Hinds [10]

> "Since the unified approach (to geometric modeling) guarantees the functional equivalence of all geometric entities of a given type, geometric operations can be performed with equal facility on simple primitives and complex sculptured geometries. Furthermore, this versatility is realized with considerable conciseness of coding : A small family of geometric-function routines accepting generic geometry inputs and yielding generic geometry outputs, forms the core of the modeler.

Our work has been concerned with using ideas from computer-aided geometric design to study mechanisms and robot manipulators. We have been using non-uniform rational B-splines (NURBS), a standard method of representing curves and surfaces in computer graphics, as a standard representation [31, 9]. Recent work [37, 20, 19] has shown that not only can static objects and swept volumes be given this representation, but that the motions themselves can be represented as NURBS with control points in the group of Euclidean motions $SE(3)$.

2. Related work.

The work outlined below takes ideas from a number of areas of robotics research, and other areas such as computer-aided design, computer graphics and topology. Much of the work of interest comes from the study of the problems of robot path-planning and collision detection.

2.1. Ideas from robotics.

Several ideas from the robotics literature have inspired and motivated the work below, in particular the robot path-planning problem has been a major source of inspiration. The canonical path-planning problem is this—given a description of the shape of a robot and constraints on how it is able to move, to design a motion of the robot which moves it from one point to another, possibly incorporating parts of a predefined trajectory on the way [25]. Many variants on this problem have been studied, including the motion of multiple robots [1, 35] and the motion of a robot amidst moving obstacles [2, 12].

Several approaches to these path-planning problems have been studied. The first work in this area approached the problems from the point of view of computational geometry—trying to calculate the computational complexity of the problems. Though much progress was made in this area (the important papers are collected in the book [16]) this work produced little in the way of practical solutions.

Two basic ideas have come to dominate work in this area since. The first is studying *configuration spaces* of mechanisms [27, 25]. The configuration space is a set which maps onto the set of all possible positions of the manipulator—in the case of robot arm manipulators the *joint space* (the space of all possible positions that the joints of the robot can take) is the most common configuration space studied. The thrust of these approaches has largely been to break down the configuration space into a number of cells, then to form a graph describing the connectivity between those cells. Much success has been had with these methods, but they suffer from the problem that a considerable amount of computation is required before even the simplest problem can be tackled. Also the configuration space grows rapidly in complexity as the number of degrees of freedom of the robot increases.

Another approach to the path-planning problem is the use of *artificial potential fields*, whereby the obstacles (either in the physical space or in a configuration space) are given a high positive potential, and the target a high negative potential [21]. Mathematical techniques for finding a potential-minimizing path are then used, which lead the robot from a start point to the target potential well. This is a very successful method for the path-planning of mobile robots, but it is less successful for manipulators as the spaces involved often have concavities which can present confusing local minima.

In order to test the unity of our model we are also studying two simpler problems. The first of these is the problem of mapping the *end-effector space* of a manipulator—that is creating a visual representation of the regions of space where the robot is able to access [5, 11] and calculating the connectivity of this space [40]. The other problem is *collision detection*—given a movement of the robot, testing whether it is able to move freely through its environment (which could include other robots or moving objects) [2].

The review papers [18, 32, 34] summarize different aspects of work on the path-planning and collision detection problems, ranging from the abstract mathematical problems suggested by this area to real-world implementations of systems designed to tackle these problems in real-world environments.

2.2. Ideas from computer graphics and geometry.

One of the basic pieces of work used below is the geometry of B-splines, which are the most common method of representing free-form curves and surfaces in computer graphics. This method of designing curves originates with the work of Pierre Bézier in developing methods for designing free-form shapes on computers for car design. Two recent enhancements (the use of rational functions so that circles, conics and quadrics can be represented exactly and the use of a piecewise polynomial scheme which allow manipulation of part of a curve while leaving other parts invariant) have produced the form which we use below, the Non-Uniform Rational B-Spline, commonly abbreviated to NURBS. Details of both the history of this and the mathematics can be found in [9].

The basic idea of all of these forms is simple. A function of an appropriate number of parameters (i.e. one for a curve, two for a surface et cetera) is created which depends on a set of point in space called the *control polygon* of the curve/surface. These points do not strictly interpolate the desired curve/surface but act as control parameters by means of which the curve/surface can be manipulated.

NURBS are used in CAD for a number of reasons. The main motivation for their development was the desire to have a free-form design scheme which allowed a designer to change the shape of curves and surfaces in an intuitive way by altering a small number of points. There also exist B-spline representations the standard shapes used in design, such as circles, conic sections, straight lines, surfaces of revolution, extruded and ruled surfaces, and these standard shapes can be blended together and combined with free-form curves and surfaces giving a smoothly blended curve/surface. These (often complicated) shapes can be stored using relatively small amounts of data.

2.3. Drawing these ideas together.

One of the major advantages of the work presented here is that it offers a unified framework for the consideration of a large class of common robotics problems. If this type of system is to be used in "real world" industry then it is essential that we have a common framework in which designers, engineers and robot programmers can work on areas of mutual interest. The ideas described in this paper also make use of CAD standards (*viz.* the use of NURBSfor free-form shape representation) in their implementation, and so can be easily and seamlessly incorporated into an existing design system in a way which makes maximal use of existing algorithms.

3. B-spline theory.

B-spline curves and surfaces are a computationally efficient method of representing free-form shapes commonly used in computer-aided geometric design. The most commonly used form of B-spline function in current CAGD system is the Non-uniform Rational B-spline (the NURBS), which allow considerable flexibility in shape design (including the exact representation of circles, conic sections, surfaces of revolution and free-form blendings between them) while retaining a large class of efficient algorithms for geometric manipulation and calculation. I shall give a brief account of B-spline theory here—more detail can be found in the books [9, 17, 31].

3.1. Mathematics of B-spline curves.

As an example the standard NURBS space curve is given by the following definition.

$$\mathbf{x}(u) = \frac{\sum_{i=0}^{n} w_i \mathbf{P}_i N_{i,p}(t)}{\sum_{i=0}^{n} w_i N_{i,p}(t)} \tag{1}$$

Where $\mathbf{P}_i$ are a set of points called *control points*, w_i are a set of weights, one corresponding to each point. By changing these weights the shape of the curve can be modified [29]. Mathematically the weights can be thought of as the fourth coordinate in a homogeneous coordinate system, defining the projection of a 4-dimensional non-rational space curve into 3-dimensional space [9]. The $N_{i,p}(t)$ are the B-spline rational basis functions, defined recursively by

$$N_{i,0}(t) = \begin{cases} 1 & \text{if} \quad t_i \leq t < t_{i+1} \quad \text{and} \quad t_i < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

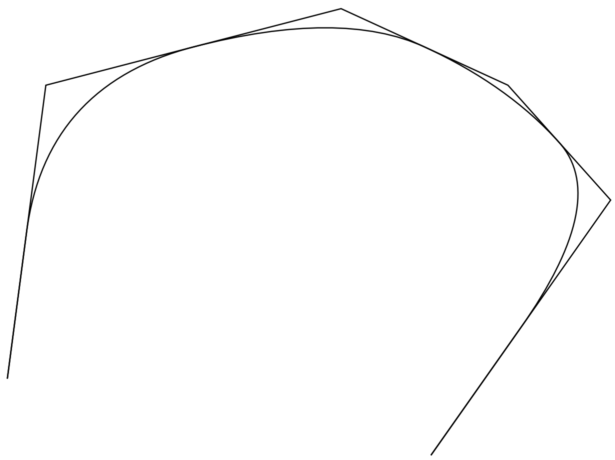

Figure 1. A B-spline curve and its control polygon.

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N(t)_{i,p-1} + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N(t)_{i+1,p-1}(t) \tag{3}$$

Here the $t_0, \ldots, t_n$ is a *non-uniform knot vector* which is a list of non-decreasing numbers, where the first and last numbers are repeated k times, where k is the order of the curve. We define p to be the degree of the curve (i.e. $p+1 = k$).

A picture of such a curve together with its control polygon is given in figure 1.

3.2. Generalizing the B-spline idea.

Below we shall use NURBS to represent a broader class of functions than the usual parameterized curves can surfaces. For this we will need to generalize the concept of a NURBS function. In this more general setting we shall define NURBS as usual but with two generalizations. Firstly we shall allow any number of parameter variables, using a tensor-product scheme similar to NURBS surfaces and volumes [23]. Secondly we will allow the control points to be in any vector space, not just the usual Euclidean two or three dimensional space. Then we can, for example, define a motion of a rigid body in space as a NURBS curve in SE(3).

A general n-variable NURBS-function has the following form (where the notation is an obvious generalization of that in section 3.1 above).

$$\mathbf{x}(u_1, \ldots, u_k) = \frac{\sum_{i_1=0}^{n_1} \cdots \sum_{i_k=0}^{n_k} \mathbf{P}_{i_1,\ldots,i_k} w_{i_1,\ldots,i_k} N_{i_1,p_1}(u_1) \ldots N_{i_k,p_k}(u_k)}{\sum_{i_1=0}^{n_1} \cdots \sum_{i_k=0}^{n_k} w_{i_1,\ldots,i_k} N_{i_1,p_1}(u_1) \ldots N_{i_k,p_k}(u_k)} \tag{4}$$

3.3. B-spline algorithms.

Perhaps the most important algorithm which can be applied to these functions is the *subdivision* algorithm. This creates two control nets corresponding to the image of the parameter space either side of a given parameter value. Below this shall be used in the implementation of divide-and-conquer strategies. Subdivision can also be seen to be *numerically stable*—basically the algorithms for subdivision are just repeated linear interpolation (see [9] for more detail).

Another idea which is used in the work below is *trimmed* NURBS [3, 33]. These arise from the problem in design where it is desired to "cut away" part of a surface, for example in producing a cut-away graphic or designing a small patch to fit part of a design. The trimmed surface is defined by defining a number of *trim curves* which are closed curves in the parameter space of the surface. Clearly these ideas can be generalized to multivariate NURBS—e.g. a surface in the parameter space of a volume can trim away part of the volume.

4. Theoretical foundations.

The basic approach of our method is this. The shape swept out by a robot or mechanism as it moves can be represented in some mathematical form. In general, these forms may analysed by using simple mathematical methods such as trigonometry and basic Euclidean geometry. However we are usually interested in the shape consisting of the area which the robot can access without touching anything else in its environment, and this shape is highly nontrivial. However we can use the techniques developed in computer-aided geometric design (NURBS, Bézier shapes et cetera) to model complicated shapes, and we will use these below to model robot workspaces.

Our work to date has focused on the analysis of open-chain mechanisms jointed together with prismatic and revolute joints—the canonical example of this being the robot manipulator arms found in industry.

In order to analyse the motion of these mechanisms from a mathematical and computational point of view we construct a number of functions describing the relationship between different aspects of the robot. These are summarized in figure 2, and described in detail in the remainder of this section.

Consider any link of the robot. Then there will be a set of *joint parameters* such as angles and extensions which specify the position of that link. We call the space of all possible values of these parameters the *configuration space*, denoted by $\mathcal{C}$. If we assign a coordinate frame to the link, we can specify its position in space as a member of the special Euclidean group $SE(3)$. The mapping from the configuration space to this group is called the *forward kinematic mapping*.

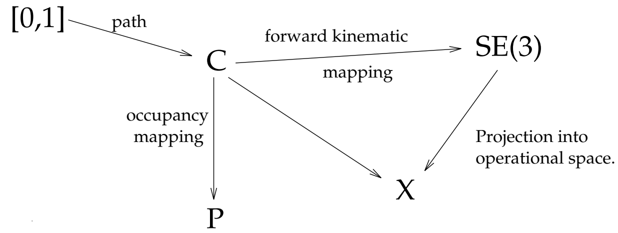

Figure 2. Diagram of kinematic functions.

Sometimes we are not interested in the position of the link as such, but of a particular tool. The set of all tool positions is called the *operational space*, and is denoted by $\mathcal{X}$. For example for a gripping tool this space is $\mathbb{R}^3 \times SO(3)$, i.e. we specify both a position and an orientation. For a rotating tool such as a polishing wheel the orientation along the axis of rotation is irrelevant, giving $\mathcal{X} = \mathbb{R}^3 \times SO(2)$, while for some tools orientation is irrelevant, giving $\mathcal{X} = \mathbb{R}^3$. The mapping from $\mathcal{C} \rightarrow \mathcal{X}$ gives different kinds of forward kinematic mappings, according to the tool used. We can calculate this mapping directly or project into $\mathcal{X}$ from $SE(3)$.

Another thing we need to formalize is the notion of when a link of the robot is interfering with an obstacle. This involves a mapping from $\mathcal{C}$ into the *physical space* $\mathcal{P}$, i.e. the three dimensional space in which the robot moves. This mapping maps from a given configuration to the surface representing that link's position in space. By subdividing configuration space and studying when its image under this mapping interferes with obstacles it is possible to break down $\mathcal{C}$ into accessible and inaccessible regions, and study its connectivity.

Finally we can specify a given motion as a mapping from the unit interval to the configuration space.

4.1. Giving kinematic functions a NURBS structure.

We can represent all of the above as generalized NURBS—generalized in the sense that there can be any number of parameters, not just curves and surfaces, and that the control points are not necessarily points in space. This allows us to represent for example volumes [23] and motions in $SE(3)$ [20].

This representation has a number of computational advantages. Firstly it is possible to represent the obstacles as NURBS, and efficient intersection algorithms exist for finding the intersection of two NURBS-structures [9]. Secondly, there exists a natural subdivision of NURBS structures, which does not exist for, say, polygons, which allows effective use to be made of divide-and-conquer strategies. Finally, the method is easily extensible to other graphical

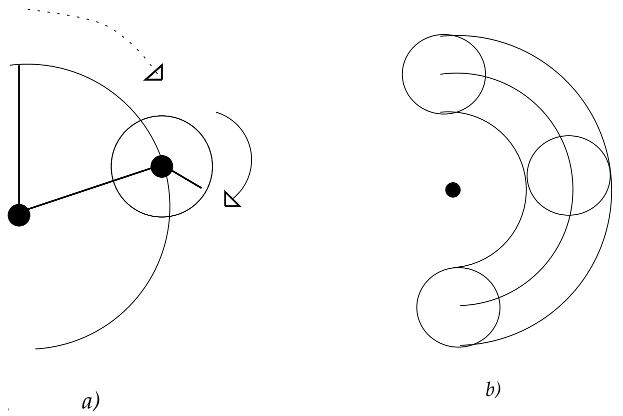

Figure 3. Motion of a 2R planar manipulator.

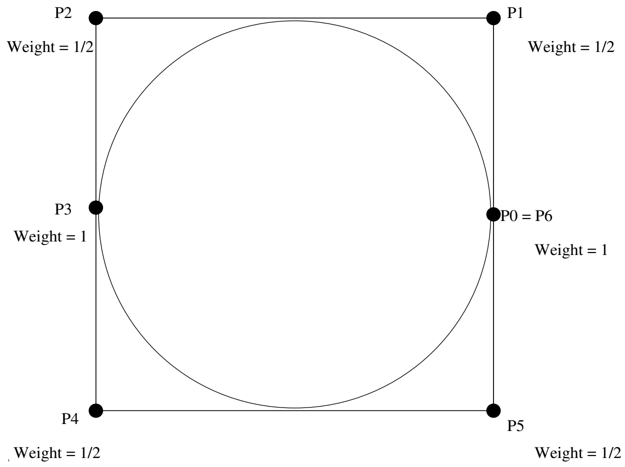

Figure 4. Construction of a NURBS circle.

representations, and many of the problems can be solved by utilizing just a subdivision algorithm and bounding-box algorithm for the obstacles. So, for example, it would be possible to introduce polygonal obstacles without much additional programming effort. This incorporates ideas from the theory of object-oriented programming—the obstacles are all derived from a "virtual class" of "geometric objects" which have only a minimal set of nontrivial properties—they can be subdivided and a bounding box can be calculated.

As an example of how a NURBS structure can be given to one of the functions above consider the occupancy mapping

$$\Omega : \mathcal{C} \times \mathcal{R} \to \mathcal{P}(\equiv \mathbb{R}^3) \tag{5}$$

which takes a configuration $c \in \mathcal{C}$ of the robot, a point $r \in \mathcal{R}$ on the robot and maps to the point in $\mathbb{R}^3$ at which r is found when the robot adopts the position specified by c.

We can construct a NURBS representation of this using ideas of volumes of revolution (see figure 3. For the first link of the arm we can construct the surface easily—we take the shape of the arm (in the picture this is a straight line, but the same technique applies to any shape which can be represented in NURBS form), and swing it around the joint between the two joint limits. There exist simple algorithms for constructing this in NURBS form. Basically we can form a circle in NURBS form (figure 4), subdivide it at the two joint limits to produce an arc, then form the a tensor-product structure whose control points are formed by taking the control points of the link and duplicating them at an appropriate place for each of the points on the arc ([31] gives details of the geometrical constructions required).

Similarly we can construct a (hyper-)volume representing the occupancy of the second link of the manipulator. Using a variant of the subdivision algorithm we can calculate the path of the joint on the end of the first link as a NURBS-curve in space. Then at each of the control points of that we construct a revolution as above, then we combine the whole into a single volume using the same technique. We can also handle mechanisms with prismatic joints using extrusions rather than revolutions.

Similar constructions can be used for as many links as is desired, producing a hierarchy of spaces representing the occupancy space of each link of the manipulator. This principle of building up kinematic functions and configuration spaces of a complicated mechanism has been called *mechanism inheritance* [13, 14, 15], and has proven a powerful tool in the study of closed-chain mechanisms. We can use these ideas of building up spaces in this way to give a NURBS structure to most of the functions illustrated in figure 2.

4.2. Interaction with obstacles.

Now we can consider what happens when we introduce an obstacle into the working environment of the manipulator (figure 5). We can make use of subdivision to get a divide-and-conquer strategy for calculating the configuration space. What we are looking for is those regions of $\mathcal{C}$ which interfere with the obstacle. More mathematically we want to find (a conservative approximation to) those $c \in \mathcal{C}$ such that for some $r \in \mathcal{R}$ the image $\Omega(c, r)$ is a point in the obstacle set Ψ.

A simple strategy for this works as follows. We construct the function Ω in NURBS form as above. We can then subdivide the image of Ω using the standard subdivision algorithms, and test these subdivided patches for interference with Ψ, using a simple bounding-box check [28]. Then if the patches do interfere with the obstacle we can subdivide them further, until we have reached the desired level of accuracy. The results can be stored in a quadtree-like data structure 5—we are currently investigating other more efficient data structures to hold the information.

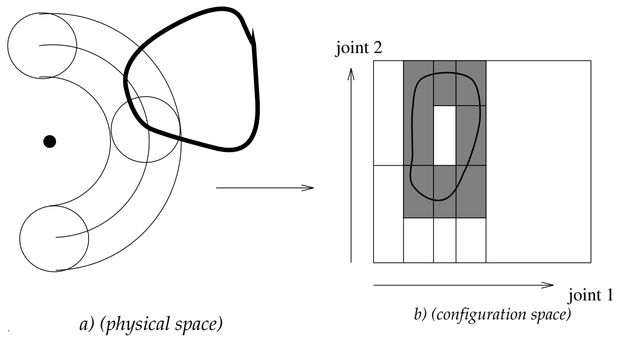

Figure 5. Interference with an obstacle.

A chief advantage of this method is that it can be very accurate in regions where it is necessary, and very rough in regions where accuracy is not required. Further, there is no predefined level of accuracy, and so, if necessary, this procedure can be repeated as often as is sensible. This ability to focus on relevant areas of the problem space is an essential requirement for an effective geometric data management system.

We are currently investigating other approaches to this problem, centered around the idea of a *trimmed* NURBS as described above in section 3.3. This would allow us to have a more accurate description of the subset of configuration space which is inaccessible to the robot.

5. Applications.

We have studied the application of these techniques to a number of well-known problems in robotics.

Firstly we can consider the relatively simple problem of constructing a model of the end-effector space—that is the image of the function Ω over all values of $c \in \mathcal{C}$ for a given point $r \in \mathcal{R}$. Once we have the function Ω in NURBS form there is a simple and computationally efficient algorithm (based on the fundamental subdivision algorithm) for taking slices of the image by fixing a parameter value.

A slightly more complicated problem involves mapping the end-effector space with parts trimmed away where the robot would hit an obstacle. We can use the method described above in section 4.2 to calculate the areas of $\mathcal{C}$ trimmed away by the obstacles, and then use a trimmed-surface visualization algorithm such as the one described in [33] to obtain the image. This is a valuable tool for deciding where to place a robot in a given environment so that it can reach a set of desired regions.

We have recently been studying the problem of collision detection using these methods. It is possible to obtain a NURBS volume corresponding to the space swept out by using functional composition (the mathematics behind this is described in [22, 7]). Basically we take a continuous NURBS

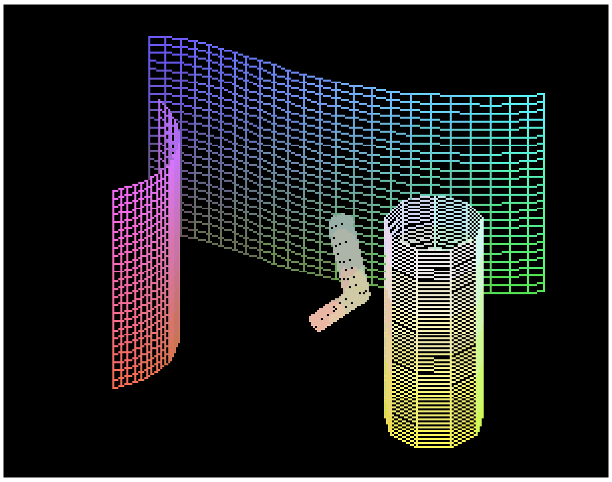

Figure 6. A frame from the simulation.

function

$$M : [0, 1] \to \mathcal{C} \tag{6}$$

describing a path in configuration space, i.e. a continuous motion of the robot parameterized by time. Then we can form the composition

$$\Omega \circ (M \times I) : [0, 1] \times \mathcal{R} \to \mathbb{R}^3 \tag{7}$$

(I is the identity map) which describes the motion of a given point on the robot ($r \in \mathcal{R}$) through time. Then we can again use standard bounding-box intersection techniques [28] to test for intersection between the image of the motion of the manipulator and the set of obstacles.

This idea can also be used for calibration—we can predetermine the path of a given point $r \in \mathcal{R}$ anywhere on the robot and then use this path as a reference for corrections based on sensor data.

Our current work is concentrating on using this model for path planning. Given a path in the configuration space, we can construct a NURBS image of this path in the physical space by using the methods described above. Several ideas concerned with path-planning are currently being investigated and compared. These include constructing a path in the configuration space and modifying it as patches become invalid, and the use of a genetic algorithm to search for optimal paths on the set of free configuration space patches. The key idea in all of this is that the searching algorithm interacts with the modelling process, so that if more detail is desired in a part of the workspace then the search algorithm can call for a greater level of subdivision in a relevant area of the space.

In order to visualize aspects of this we have used the computer program *Maple V* to produce simple animations. A frame from one of these animations is shown in figure 6.

6. Future Work.

This paper has presented a snapshot of a research project in progress, and there are several ways in which this work will be extended in the future.

Firstly in order to be a practical robot programming system the ideas developed above will have to be extended to be part of a larger CAD system, and will also have to incorporate a more elaborate programming language. It is desired that this programming system should be largely visual, allowing the user to move the robot in the simulated workspace and test out the programs being developed using simulated input.

The development of a robot programming environment leads to the desire to incorporate sensor input into the system. This could be at various levels. It would be a relatively simple task to incorporate responses to basic range finding and on/off sensors. At the other end of the scale the use of NURBS for visual image reconstruction [38, 24] offers a realm of possibility for using vision as part of the feedback process.

One area which we have only begun to explore is the possibility of using these techniques to analyse the motion of robots in an environment which has other moving components. There are three ideas at work here. Conceptually the simplest is the motion of a mechanism in a space which is cluttered with objects moving on predefined trajectories—we want to know whether the objects collide, if so, when, and how we can plan a path for the manipulator to move around the objects.

Another aspect is robots making coordinated motions, for example two manipulators which need to work in close proximity or which need to collaborate on a task. Using the techniques described above it should be possible to trim away the free space of one robot by the motion of the other, or else to calculate the mutual configuration space of the two robots.

The natural mathematical framework for studying these moving obstacles is to extrude the moving obstacles into a four-dimensional space-time [2, 4] and study the intersections there. Figure 7 illustrates this idea by extruding two-dimensional objects into three-dimensional space-time. The framework of generalized NURBS described above in section 3.2, together with the algorithms of NURBS-skinning and extrusion [36, 30] provide a powerful set of tools for tackling these hard problems.

The final aspect of moving obstacles is that a robot can move obstacles itself, by use of a gripper or similar. We are intending therefore to incorporate into our system means by which the robot can be considered to have picked up another object at a point in its trajectory, and to recalculate the swept volumes accordingly for the new "robot + payload" mechanism.

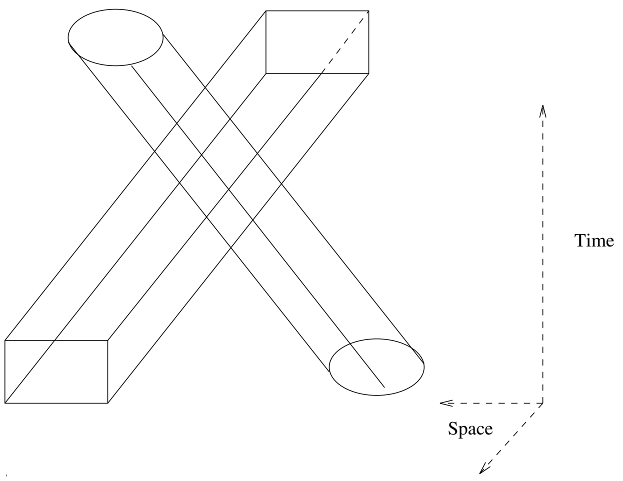

Figure 7. Two-dimensional objects extruded into three-dimensional space-time.

Another way in which we can extend this work is to model a wider class of mechanical systems, leading to an integrated CAD system for the design of mechanical and mechatronic devices. This could incorporate both the ideas above and other recent work on the theory of mechanisms with closed-loops [13, 14, 15].

Finally there are a number of ways in which the computational aspects of the project could be enhanced. We are currently studying different data structures to hold the information about which patches are free and which blocked. In the future we are looking to use parallel computers to speed up some of the algorithms—many of the B-spline algorithms, especially those concerned with subdivision and calculating intersections are very amenable to parallelization. Also some graphical computer workstations use dedicated hardware to perform some of the standard algorithms, this also offers exciting possibilities for future development.

7. Conclusions.

In this paper we have presented the basic mathematical and computational framework for a new method of modelling robot manipulator workspaces. This method is computationally efficient and is compatible with CAD standards, which will allow the development of CAD systems which can incorporate robots and mechanical systems on an intuitive level, making use of the powerful algorithms which have been developed in that field. We have shown that this method can provide a unifying framework for approaching a large number of geometrical problems in robotics, and in the future we hope to extend the theory and develop computer simulations which demonstrate the extended validity of this type of modelling in an even wider range of situations.

References

[1] P. Alison, M. Gilmartin, and P. Urwin. Strategic collision avoidance of two robot arms in the same work cell. In A. Lenarčič and B.B.Ravani, editors, *Advances in Robot Kinematics and Computational Geometry*, pages 467–476. Kluwer, 1994.

[2] S. Cameron. Using space-time for collision detection : solving the general case. In Warwick [39], pages 403–415.

[3] M. S. Casale. Free-form solid modeling with trimmed surface patches. *IEEE Computer Graphics and Applications*, pages 33–43, January 1987.

[4] H. H. Cheng. Real-time four-dimensional collsion detection for an industrial robot manipulator. In *Proceedings of the 3rd National Conference on Applied Mechanisms and Robotics (Cincinatti, Ohio)*, volume 1, pages 1–13, 1993.

[5] S. Chiaverini and C. Vicinanza. Reachable workspace computation for planar revolute jointed arms. In Warwick [39], pages 93–105.

[6] J. Craig. *Introduction to Robotics*. Addison-Wesley, second edition, 1989.

[7] T. D. DeRose, R. N. Goodman, H. Haken, and S. Mann. Functional composition algorithms via blossoming. *ACM Transactions on Graphics*, 12(2):113–135, April 1993.

[8] Editorial. *IEEE Robotics and Automation Magazine*, 1(1), March 1994. front cover picture and associated comment, work from Sandia National Laboratories.

[9] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, third edition, 1993.

[10] R. T. Farouki and J. K. Hinds. A hierarchy of geometric forms. *IEEE Computer Graphics and Applications*, pages 51–78, May 1985.

[11] R. Featherstone. A hierarchical representation of the space occupancy of a robot mechanism. In J.-P. Merlet and B. Ravani, editors, *Computational Kinematics (INRIA, September 1995)*. Kluwer, 1995.

[12] K. Fujimura and H. Samet. A hierarchical strategy for path-planning among moving obstacles. *IEEE Transactions on Robotics and Automation*, 5(1), February 1989.

[13] C. Gibson and D. Marsh. Concerning cranks and rockers. *Mechanism and Machine Theory*, 23(5):355–360, 1988.

[14] C. Gibson and D. Marsh. On the linkage varieties of the Watt 6-bar mechanisms. *Mechanism and Machine Theory*, 24(2):106–126, 1989.

[15] C. Gibson and D. Marsh. On the geometry of geared 5-bar motion. *Journal of Mechanical Design*, 112(4):620–627, 1990.

[16] J. Hopcroft, J. Schwartz, and M. Sharir, editors. *Planning, Geometry and Complexity of Robot Motion*. Ablex, Norwood, N.J., 1987.

[17] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A.K. Peters, 1989.

[18] Y. Hwang and N. Ahuja. Gross motion planning—a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[19] B. Jüttler. Spatial rational motions, March 1996. Seminar, University of Dundee.

[20] B. Jüttler and M. G. Wagner. Computer-aided design with spatial rational B-spline motions. Technical report, University of Dundee / University of California, Davis, December 1995. To appear in ASME Journal of Mechanical Design.

[21] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[22] D. Lasser. Composition of tensor product Bézier representations. *Computing Supplementum*, 8:155–172, 1993.

[23] D. Lasser. Rational tensor product Bézier volumes. *Computers and Mathematics with Applications*, 28(8):49–62, 1994.

[24] S. Lavallée and P. Szeliski. Recovering the position and orientation of free-form objects from image contours using 3D distance maps. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 17(4):378–390, 1995.

[25] T. Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE transactions on systems, man and cybernetics*, SMC-11:681–698, October 1981.

[26] T. Lozano-Pérez. A simple motion-planning algorithm for general robotic manipulators. *IEEE Journal on Robotics and Automation*, RA-3(3):224–238, 1987.

[27] T. Lozano-Pérez and M. Wesley. An algorithm for plannign collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22:560–570, October 1979.

[28] Q. Peng. An algorithm for finding the intersection lines between two B-spline circles. *Computer Aided Design*, 16(4), July 1984.

[29] L. Piegl. Modifying the shape of rational B-splines. part 1 : curves. *Computer Aided Design*, 21(8):509–518, 1989.

[30] L. Piegl and W. Tiller. Curve and surface constructions using rational B-splines. *Computer Aided Design*, 19(7):485–498, 1987.

[31] L. Piegl and W. Tiller. *The $\mathcal{NURBS}$ Book*. Springer, 1995.

[32] J. Schwartz and M. Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence*, 37:157–169, 1988.

[33] M. Shantz and S.-L. Chang. Rendering trimmed NURBS with adaptive forward differencing. *Computer Graphics*, 22(4):189–198, August 1988.

[34] M. Sharir. Algorithmic motion planning in robotics. *Computer*, 22:9–20, March 1989.

[35] Y. Shin and Z. Bien. Collision-free trajectory planning for two robot arms. *Robotica*, 7:205–212, 1989.

[36] W. Tiller. Rational B-splines for curve and surface respresentation. *IEEE Computer Graphics and Applications*, pages 61–69, September 1983.

[37] M. G. Wagner. Planar rational B-spline motions. *Computer-Aided Design*, 27(2):129–137, February 1995.

[38] Y. Wang and J. Wang. On 3D model construction by fusing heterogeneous sensor data. *CVGIP-Image Understanding*, 60(2):210–229, 1994.

[39] K. Warwick, editor. *Robotics, Applied Mathematics and Computational Aspects*. Clarendon/IMA, 1993.

[40] P. Wenger and P. Chedmail. Ability of a robot to travel through its free work space in an environment with obstacles. *International Journal of Robotics Research*, 10(3):214–227, 1991.

From the Proceedings of the IEEE International Joint Symposia on Intelligence and Systems, Washington DC, November 1996.