A CAD Representation of Robot Manipulator Workspace.

Colin G. Johnson.
Department of Computer Science,
University of Exeter,
Exeter, EX4 4PT, England, U.K.

and

Duncan Marsh.
Department of Mathematics,
Napier University, 219 Colinton Road,
Edinburgh, EH14 1DJ, Scotland, U.K.

Abstract: If robot programming is to advance to the stage where the well-studied problems of automated path-planning, collision detection, workspace mapping et cetera can leave the research laboratory and take their place in industry, then there is a need for a new kind of robot programming. Incorporating robot programming into a CAD system is one possibility, and the first stage in producing such a system would be to represent robot workspaces in a CAD-amenable format. This paper describes a representation of robot manipulator workspaces in terms of non-uniform rational B-splines (NURBS), a standard representation for free form shapes in CAD. Applications of the technique to collision detection, path planning and workspace visualization problems are outlined. The paper concludes by providing signposts towards a new model of mechanism programming which is grounded in CAD free-form modelling concepts.

1 Introduction

Currently robot manipulators (the robot "arms" commonly used in industry) are typically programmed in one of two ways. In the first form a robot is led through a sequence of motions using a hand-held controller commonly referred to as a *teach pendant*. The second is to write programs in a computer language (either a specialist language or a library added to an existing language) and test the results using a graphical simulator.

What we would like to explore in this paper are the foundational pieces of mathematical and computational modelling which pave the way for a new approach to robot task preparation. The new approach applies graphical programming techniques within the context of a computer-aided geometric design system. In this paper we develop a new model for robot *workspace*—the space in which the robot is constrained by its geometry to move—based on extensions of of non-uniform rational B-splines (NURBS), the most commonly used mathematical representation of free-form shapes in CAD systems. It shall be shown that NURBS functions offer a mathematically and computationally powerful representation of robot workspaces through which algorithms, developed for other reasons in CAGD (Computer Aided *Geometric* Design) and computer graphics, can be brought to bear in providing a unified approach to a number of problems in robotics. This facilitates the seamless integration into CAD systems of geometrical problems which have been well studied in the past, allowing them to be incorporated into existing industrial design practice.

The eventual development of new approaches to robot programming based on these ideas offers a number of advantages over existing methods. Firstly the robot can be programmed mainly offline, thus liberating it for use whilst other tasks are being prepared. Secondly these approaches place robot programming in a familiar context—the CAD system—thus empowering designers to program robots at the *task level*, and allowing the technical detail about the robot's capabilities to be built into the programming system. Thirdly this approach does not require the programmer to have a detailed knowledge of robot engineering. Instead, the detail is embedded into the machine thus allowing the programmer to concentrate on task design.

This paper concentrates upon the CAD foundations of our model. Details of the application of these ideas can be found

in [9, 10].

2 Algorithms for workspace generation

The foundations for our CAD system for robot programming shall be based on a modelling technique which uses NURBS for the representation of robot workspaces. In this section we give a brief introduction to mechanism kinematics, the describe two NURBS constructions. The first gives a multivariate NURBS function for the whole robot workspace, while the second generates the volume swept out when a robot follows a particular trajectory.
This section assumes a basic knowledge of geometrical methods in CAD and the geometry of NURBS, details of which can be found in many books such as [7, 17].

2.1 Mechanism kinematics

Almost all robot arms in use in industry have an *open-chain* kinematic structure. Such mechanisms consist of a chain of *links* connected with *joints*. These joints are either *revolute joints* which rotate around an axis, or *prismatic joints* which move along an axis.
In order to specify the geometry and kinematics of such a robot needs three items of information. Firstly, the spatial position/orientation of the robot is given by a cartesian coordinate frame. Secondly, the physical geometry of the links is given by by a NURBS surface $\mathbf{S}_l$ for each link l (it is simple to extend this to several surfaces per link). Finally, we specify how the links are connected together, using the Denavit-Hartenberg notation—the standard notation used in kinematics [4, 5]. We describe this briefly as follows (see figure 1). We begin by taking a line ℓ_i through the axis of each joint of the mechanism, i.e. the axis that a link either rotates around (revolute joint) or slides along (prismatic joint). Each pair ℓ_i, ℓ_{i+1} is joined by their unique common perpendicular (unless they are parallel, in which case any common perpendicular will suffice). Next we specify the kinematic relationship between these links exactly using four parameters. Two of these parameters, the *link length* a_{i-1} and the *link twist* α_{i-1} specify the fixed relationship between the two axes forced by the physical link. The remaining two, the *link offset* d_i (which is variable for a prismatic joint) and the *joint angle* θ_i (which is variable for a revolute joint) specify the relationship between two adjacent links.

2.2 Workspace generation

We use the Denavit-Hartenberg specification to generate a set of mappings $\omega_i : \mathbb{R}^2 \times \mathbb{R}^i \to \mathbb{R}^3$, where i ranges from $1, \ldots, d$, where d is the number of degrees of freedom of the robot. The function ω_i specifies the region of space occupied by the robot in a particular position. Consider the mapping $\omega_i : (u, v) \times (r_1, \ldots, r_i) \mapsto (x, y, z)$. This takes a value of the parameters (u, v) which specify a point in the domain of $\mathbf{S}_i(u, v)$, and $(r_1, \ldots, r_i)(j = 1, \ldots, i)$ which specify the values of d_j (when the jth link is prismatic) or θ_j (when the jth link is revolute). The image (x, y, z) is a point in $\mathbb{R}^3$ which specifies where the point $\mathbf{S}_i(u, v)$ is found when the robot is in the position specified by $(r_1, \ldots, r_i)$.
The next stage is to give these mappings are given a NURBS structure. Place the control net for the surface $\mathbf{S}_1(u, v)$ in base position. Form the tensor product of $\mathbf{S}_1(u, v)$ with a motion-curve $C(r_1)$, which is an arc (ranging between the upper and lower limits of θ_1) of a unit circle around the base-axis if the joint 1 is revolute, and a NURBS line (ranging between the upper and lower limits of d_1) along the axis if the joint is prismatic.
The penultimate part of the algorithm is to give a basic structure on which to place the surface making up link 2. To do this a NURBS arc/line segment $D(r_1)$ is constructed having a radius/length a_{i-1}. The control net $\mathbf{S}_2(u, v)$ is placed at the base position, and then displaced by a_{i-1}, rotated by α_{i-1}, and finally rotated/translated by whichever of θ_i and d_i is fixed. Finally a tensor product between the line/arc $C(r_2)$ and the transformed $\mathbf{S}_2$ is formed, and a further tensor-product with $D(r_1)$ gives the 4-variable NURBS function $\omega_2(r_1, r_2, u, v)$. We repeat this process until the occupancy functions $\omega_n(r_1, \ldots, r_n, u, v)$ for all links have been generated.

2.3 Modelling specific motions

In addition to the mappings for workspace generation we define mappings which give a NURBS model of the mapping which defines volume of space (or space-time) occupied by the robot during the execution of a given trajectory. More precisely for a given motion $\mathbf{M}$, specified as a NURBS path in configuration space [14], we define a function for each link $\Omega_i(M) : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^3$. This function takes a pair (u, v) specifying a point in the domain of the $\mathbf{S}_i(u, v)$ and a parameter t specifying the distance travelled along the motion $\mathbf{M}$, to obtain $\Omega_i(\mathbf{M}) : (u, v) \times t \mapsto (x, y, z)$, where (x, y, z) is the point occupied by the image of $\mathbf{S}_i(u, v)$ when the link is at the point in configuration space given by $\mathbf{M}(t)$.
We model this using a *geometric swept-volume algorithm*. This takes a template surface $\mathbf{S}(x, y)$ and moves it along a trajectory $\mathbf{T}(t)$ whilst also executing a local motion of the surface, producing a swept volume $\mathbf{V}(x, y, t)$ in space. We can express it thus

$$\mathbf{V}(x, y, t) = \mathbf{T}(t) + N(\mathbf{S}(x, y), t)$$

where N is a transformation of the surface with respect to its fixed position, which varies with changes in t. This has been used in planar kinematics, where the motion N was a multiplication of the control net of $\mathbf{S}(x, y)$ by control points in a of transformation matrices [19]. This allows us to calculate the volume swept out when we move the surface through space whilst simultaneously transforming the shape with respect to a moving coordinate frame.

We calculate these $\Omega_i(M)$ in two stages. In the first stage we take the link-surface $\mathbf{S}_i(u, v)$, placed with respect to a coordinate frame at the origin. If the ith joint is a revolute joint, we form a volume of revolution $\mathbf{V_i}(u, v, t)$ by forming a tensor product of S_i with an arc of a circle in NURBS form [17]. Similarly for a prismatic joint we tensor product the surface with a straight line along the axis to form a volume of extrusion. The length and parameterization of these lines and arcs are derived by reparameterizing a standard NURBS circle or line by a function specifying the motion of the link.

For the first joint this volume is $\Omega_1(M)$, the space swept out by S_1 as the first joint moves around a fixed coordinate frame. However for the other joints the axis itself is moving, so we have a second stage. Take a point at the ith joint and apply the rotation/extrusion to that, giving a NURBS-curve $\mathbf{T}(t)$ in space, having a knot vector which we shall call U_t. Then use degree-raising and knot insertion to equate U_t with U_r, the knot-vector of $\mathbf{V_i}(u, v, t)$ in the t direction. This produces a new set of control points for $\mathbf{T}(t)$ which we call $(T_0, \ldots, T_{n_t})$. Create a set of new points P_{ijk} from the control points V_{ijk} of $\mathbf{V}(u, v, t)$ and the control points T_i of $\mathbf{T}(t)$.

$$P_{ijk} = RV_{ijk} + T_i$$

Where R is the rotation matrix (an affine transformation) that carries the frame based at the origin into the Frenet frame moving along the curve. The desired swept surface $\mathbf{V}(u, v, t)$ is defined by

$$\mathbf{V}(u, v, t) = \sum_{k=0}^{n_t} \sum_{j=0}^{n_u} \sum_{i=0}^{n_v} P_{ijk} R_{i,d_u}(u) R'_{j,d_v}(v) R''_{k,d_t}(t)$$

Where $R_{i,d_u}(u)$, $R'_{j,d_v}(v)$ and R''_{k,d_t} are the non-uniform rational basis functions defined over U_u, U_v (the knot vectors of $\mathbf{S}(u, v)$) and U_t respectively.

Note that isoparametric surfaces for fixed t values of $\mathbf{V}(u, v, t)$ consist of $\mathbf{S}(u, v)$ transformed to an appropriate position along the curve. That the placement of these surfaces at the control points is sufficient to describe the entire motion follows from the the affine invariance property of B-splines [7].

2.4 Comments

This representation restricts the motions allowed to those which can be represented in NURBS form. It could be well argued that this is not a *restriction* at all. Firstly, we can approximate any motion as accurately as desired using a NURBS path. Secondly, we have to design a motion using *something*, and NURBS, with their properties of local control, control over their smoothness and their ability to incorporate many other kinds of motion such as straight-line interpolants and circles [16] offer an intuitive and geometrically elegant method for this.

It can be seen that this can be extended to the case of creating a swept volume in four-dimensional space-time [3]. This is important for studying the interaction of a robot with other moving obstacles [1, 2, 3, 6], or attempting to detect self-intersections.

The main advantage of this representation is that it allows the motion, the shape of the links and the resultant swept volume to be represented in a common form, and has the added advantage that that form is a standard in CAD. Such advantages are not to be found in other swept-volume models of workspace such as [13].

3 Applications

The workspace representations developed above yields a CAD framework in which robot programming can be incorporated into a CAD system. Thus it is possible to develop systems which work at a higher level of automation then at present, based around a *task planning* ideal, where the gross tasks are specified in a graphical or natural language and the fine details formulated within the software.

In working towards this aim a number of problems arise which can be abstracted from the task. The first of these is the development of a *collision detection* algorithm, that is given a motion within a (designed) environment, determining whether the robot hits anything in its environment. Cameron [1, 2] has identified four ways in which potential collision can be detected, from which we have chosen to use testing for the intersections of the robot's swept volume with other static object in $\mathbb{R}^3$. For situations which are time-dependent we use another method, the intersection of the four-dimensional space-*time* sweep, to test for self-intersections or for studying motions in dynamic environments.

The key to our algorithms is the *convex hull property* of NURBS. We have used this to create bounding boxes, and use generalizations of surface-surface intersection algorithms [15, 18, 21] to both check *whether* there is a collision, and if there is to check *where* the collision occurs.

Harder problems are *accessibility checking* and *path planning*. This is part of developing a higher level of automa-

tion, where instead of using the computer to test paths that have been designed, we merely need to specify the general constraints and task-requirements, and require the program decide whether any path is feasible, and if so, to find it. Clearly this is a hard problem, and while much research has been carried out, it has yet to find its way into industrial practice.

It is our intention that by embedding these problems within a CAD framework these methods will become more accessible to industry. We have looked at two approaches to these problems. One approach is to take advantage of the fast calculation times for collision detection and develop a test-and-correct system. This could take ideas from *genetic algorithms*, taking a large number of simple paths and finding a good path by iterating through selecting the best current paths and then subdividing and recombining them until a good solution is found.

Our work to date has concentrated on a second, more geometric approach, where models of the whole workspace are prepared, as described in section 2 above, and regions which cannot be accessed are trimmed away. This begins by cutting away large areas of the workspace which cannot be reached by gross motions of the earlier links, so that by the time we are looking at the fine detail we are able to concentrate on small, relevant regions. This combination of fine detail where it is needed combined with cruder chopping away of undesirable regions makes for a combination of speed and accuracy which previous approaches have not obtained.

A further problem with a geometrical flavour is *workspace visualization*—that is producing a graphic image of the entire space which can be reached by the robot. This is where the natural graphical nature of our method had a clear advantage—we can take the mappings of the occupancy space, possibly trimmed away as above, and render the images of these functions using rapid subdivision methods [7, 8, 11].

So far we have considered work within a designed environment. At present we are working towards incorporating *computer vision* to enable these methods to be applied within an unknown environment. This could draw on the work of Wang and Wang [20], who use the projection of a *structured light* pattern onto objects to recover curvature information, and Lavallée and Szeliski [12], who use range data to recover B-spline models of unknown surfaces.

Details of algorithms for several of these applications are given in our other papers [9, 10].

References

[1] S. Cameron. A study of the clash-detection problem in robotics. In *IEEE International Conference on Robotics and Automation*, pages 488–493. IEEE Press, March 1985.

[2] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation*, 6(3), June 1990.

[3] S. Cameron. Using space-time for collision detection : solving the general case. In K. Warwick, editor, *Robotics, Applied Mathematics and Computational Aspects*, pages 403–415. Clarendon/IMA, 1993.

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

[5] J. Denavit and R. Hartenberg. A kinematics notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics (Transactions of the ASME)*, June 1955.

[6] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1419–1424, 1986.

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

[8] M. Frühaud and M. Göbel, editors. *Visualisierung von Volumendaten*. Springer, 1994.

[9] C. G. Johnson and D. Marsh. Modelling robot manipulators in a CAD environment using B-splines. In N. Bourbakis, editor, *Proceedings of the IEEE International Joint Symposia on Intelligence and Systems*, pages 194–201. IEEE Press, 1996.

[10] C. G. Johnson and D. Marsh. A robot programming environment based on free-form CAD modelling. IEEE International Conference on Robotics and Automation, Leuven, Belgium, 1998.

[11] D. Lasser. Free-form volumes : Definitions, applications, visualization techniques. Technical Report Interner Bericht 238/94, Universität Kaiserlautern, Fachbereich Informatik, 1994. Habilitationschrift.

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

[13] Z.-K. Ling and Z.-J. Hu. Use of swept volumes in the design of interference free spatial mechanisms. *Mechanism and Machine Theory*, 32(4):459–476, 1997.

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

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

[16] L. Piegl. On NURBS : A survey. *IEEE Computer Graphics and Applications*, January 1991.

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

[18] T. W. Sederberg and S. R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, January/February 1986.

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

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

[21] J. Yen, S. Sprach, M. Smith, and R. Pulleyblank. Parallel boxing in B-spline intersection. *IEEE Computer Graphics and Applications*, pages 72–79, January 1991.