PERSONNEL OCCUPIED WOVEN

ENVELOPE ROBOT

POWER

SEPTEMBER 30, 1988

FINAL REPORT

DR. F. C. WESSLING

Submitted in response to:

Grant #NAGW-847

A joint project of:

The Johnson Research Center (UAH)

Pace & Waite, Inc.

and

Wyle Laboratories

Submitted to: Dr. Jeffery D. Rosendahl, Assistant Administrator
Space Science and Applications (Science) Code E
National Areonautics and Space Administration
600 Independence Avenue
Washington, D. C. 20546

ACKNOWLEDGMENT

The principal investigator wishes to recognize the following individuals and organizations for their contributions to this work.

The University of Alabama in Huntsville

M. Hagadorn	Controls and drawings
M. C. Ziemke	Failure Modes and Effects
W. Teoh	Controls
A. Choudry	Simulation Director
V. Harrand	Computer Simulation
P. Janssen	Computer Simulation

Wyle Laboratories

G. B. Bakken	Structures and Materials
V. Patel	Structures and Materials

Pace and Waite, Inc.

R. Pace	Subcontract Manager
S. Reinartz	Systems Review and Report
S. Smith	Power and Energy Requirements
M. Bangham	ECLSS
R. Heckman	Crew Compatability Assessment

PERSONNEL OCCUPIED WOVEN

ENVELOPE ROBOT

POWER

FINAL REPORT

The Personnel Occupied Woven Envelope Robot (POWER) concept has evolved over the course of the study. Many potential applications were envisioned. See Table I.

TABLE I

Potential Applications of POWER

1. Changing out and servicing payloads on the Power Tower payload platform.
2. Maintaining subsystems such as propulsion and attitude control.
3. Providing satellite and free flyer service.
4. Performing inspections.
5. Supporting the man tended option.
6. Performing remote control operations for hazardous duty.
7. Capturing satellites during final approach.
8. Docking for the orbiter, the orbital maneuvering vehicle and the orbital transfer vehicle.

The original concept utilized the use of a flexible tunnel as a structural element. See Figure 1 for the proposal concept of POWER. A careful analysis of the advantages and disadvantages of this structural element led to a design with much better structural integrity as shown in Figure 2. The stack of Stewart tables makes a stable flexible mechanism. A complete description of the structural considerations were presented in the progress report dated June 1, 1986.

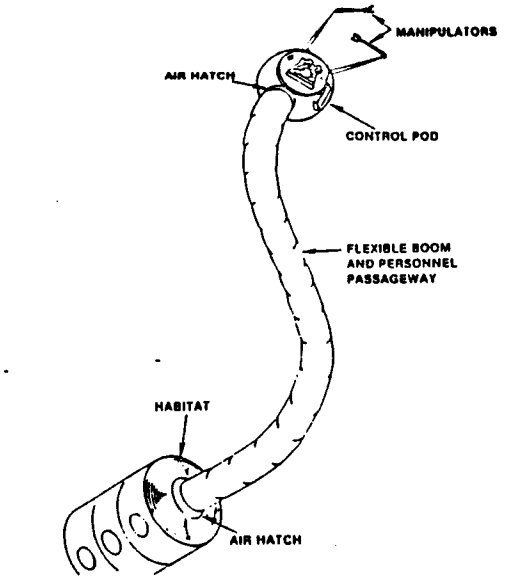


Figure 1. Proposal Concept of POWER

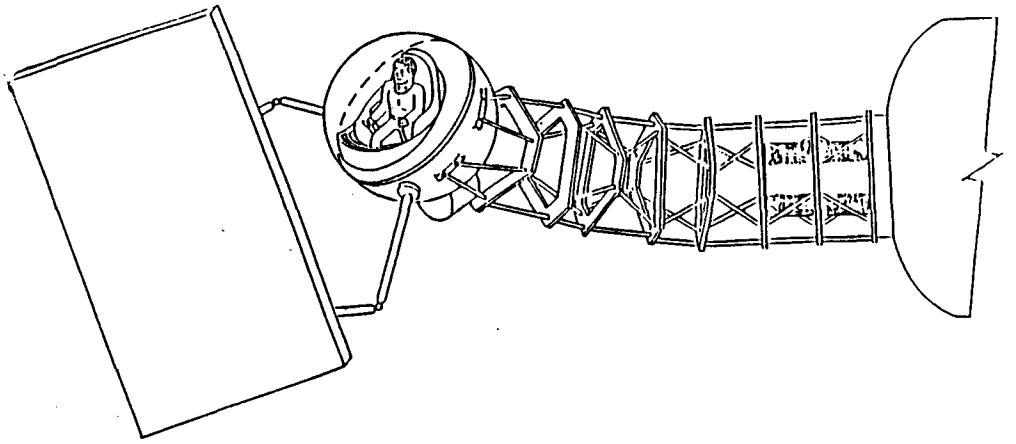


Figure 2. Human Occupied Space Teleoperator (HOST)

An analysis of the tunnel structure and operation showed that a system option designed without the tunnel appeared to have certain advantages. One no longer needed to provide micrometeriod protection for the tunnel. Air loss due to tunnel permeability was avoided and energy used to provide tunnel motion was no longer required. Such a system would have the pod dock directly with a module or node port and an astronaut could pass from the Space Station directly to the pod. The doors of the pod could remain open when the pod was docked so that the Space Station environmental control and life support system (ECLSS) would maintain the atmosphere in the pod. The pod would have its own ECLSS for normal pod operations. This concept was explained in a progress report dated November 30, 1987. The system was subsequently unofficially renamed Host (Human Occupied Space Teleoperator). See Figure 3.

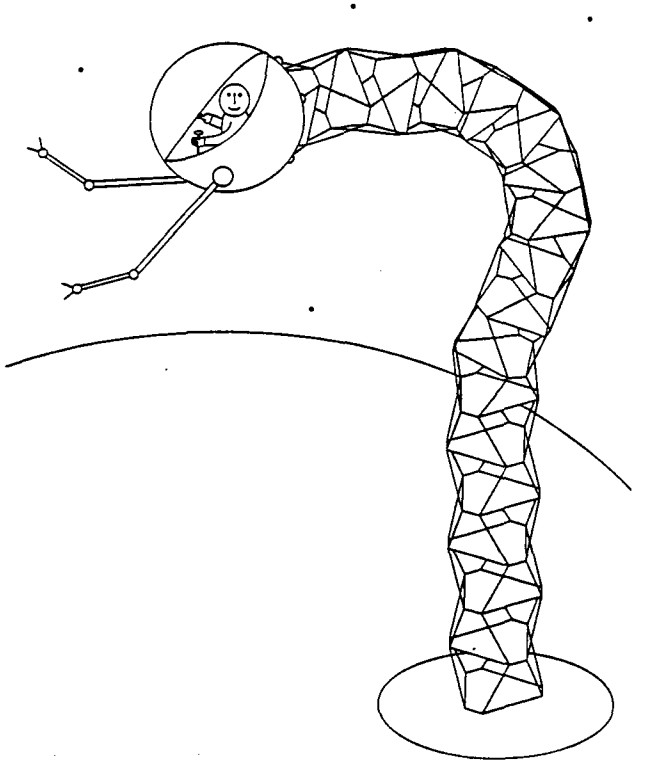


Figure 3. HOST With No Tunnel

This combination of the stacked tables and the pod (HOST) could also be used to build Space Station. HOST was sized to fit in the shuttle cargo bay. Consequently, HOST could be carried into orbit and used to assemble pieces of the truss which were transported to orbit earlier. The astronaut could transfer from the orbiter to the pod through the orbiter airlock. He would then operate HOST similar to a cherry picker used by power line repairmen on earth. See Figure 4. He would, however, have the added advantage of the RMS grappler and the robotic arms to assist him. The astronaut could disengage HOST from the orbiter once a sufficient amount of the truss were assembled. This could be accomplished by using the RMS grappler on the pod and the pod's arms as a temporary attachment to the truss until the base of the stacked tables were swung over to a truss mount and attached. Then, the astronaut could use the truss as a base of operations and use HOST to unload the orbiter payload bay on subsequent trips. Re-entry of the astronaut to the orbiter could be accomplished by moving the pod back to the orbiter. The astronaut could then leave the pod and disengage the pod from the airlock. Thus, HOST would be available for use on subsequent trips, and would remain in orbit.

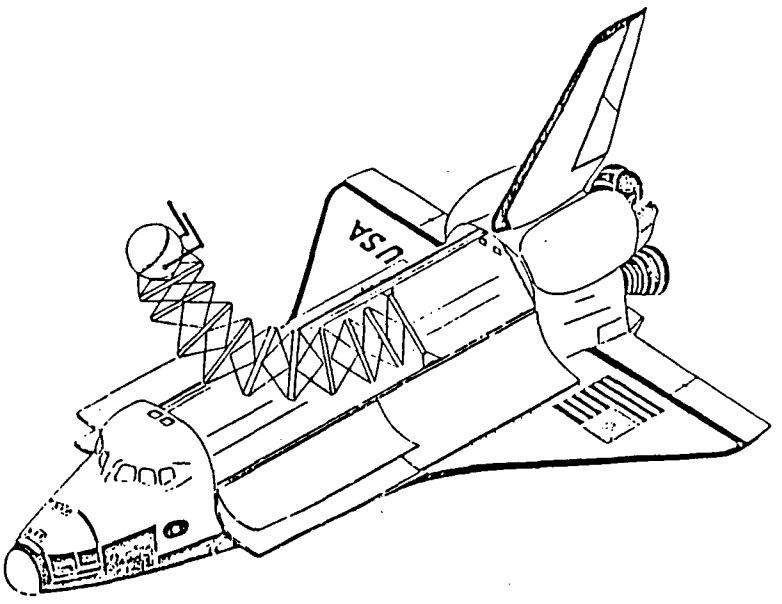


Figure 4. HOST Deployed From Orbiter

Although the tunnel may not be necessary for operation of the pod, a combination of stacked Stewart tables and the tunnel would serve as a docking mechanism (jet-way) for the shuttle at the Space Station. This could allow misalignment between the Space Station and the shuttle to be accommodated by the tables, while at the same time provide a pressurized passageway for the astronauts. See Figure 5.

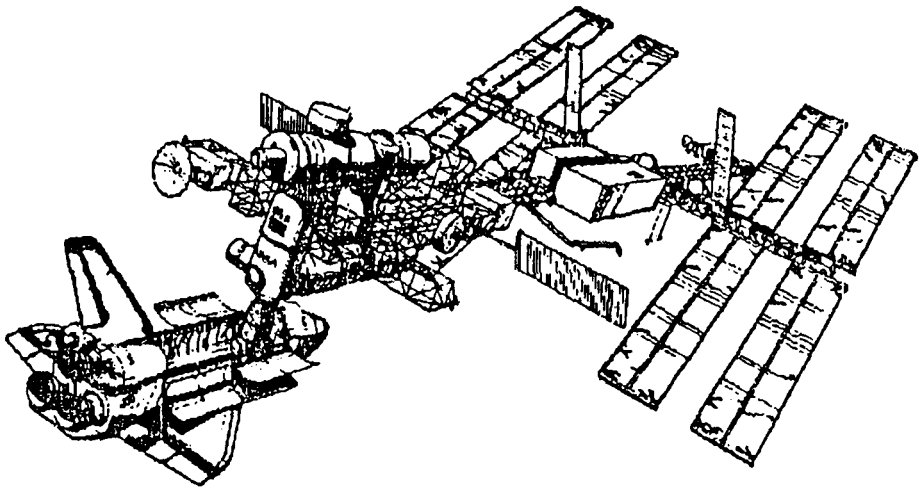


Figure 5. Orbiter Docking to Host Flexible Tunnel

Thus, crew transfer between the shuttle and the Space Station would be easily accommodated.

A failure modes and effects analysis was presented in a progress report dated June 1, 1987. Specific failure mode probability would need to be performed during hardware design. Five major systems or elements were considered: 1) base airlock hatch, 2) personnel access tunnel, 3) segment base truss structures, 4) segment actuator, and 5) control pod. The conclusion of the analysis was that further study of the failure modes and effects was necessary during specific hardware design, but that no insurmountable difficulties were encountered.

An accommodations assessment showed that HOST could be designed to fit into the payload bay, and that structural attachment at both ends and several places in between would be required for transport. No unusual orbiter services or accommodations would be required.

Accommodations on Space Station showed the primary interface would be the pod attached to a station resource node once the Station was built. The base of the extension truss could be mounted to the Station structure or to a rail that the mobile servicer would use. Standard Space Station services would be used except that 28 volt direct current might be used to drive the actuators; more electrical trade-off studies need to be done. Particular attention needs to be paid to the power consumption of the actuators and of the electronics. Both of these elements were conservatively rated for the purpose of this study. Yet, this study uncovered no major difficulties.

A considerable effort was expended on modeling and simulating HOST on a Silicon Graphics Integrated Raster Imaging System (IRIS) work station. A detailed description of the solid modeling work is given in Appendix A, one of the computer generated solid model images is shown in Figure 6. This shows the edge of the Space Station module, the micrometeoroid protection canister for the

tunnel, the stack of Stewart tables, and the pod. A detailed derivation of the equations used in the kinematic studies is given in Appendix B, one of the computer generated images displayed on the IRIS work station is shown in Figure 7. This shows the solid model of the Space Station with a stack of Stewart tables attached. The stack motion is controlled by keyboard input and a computer "mouse". The numbers on the screen give the position and orientation of the pod, and the lengths of the actuators. A brief video tape of the work is available.

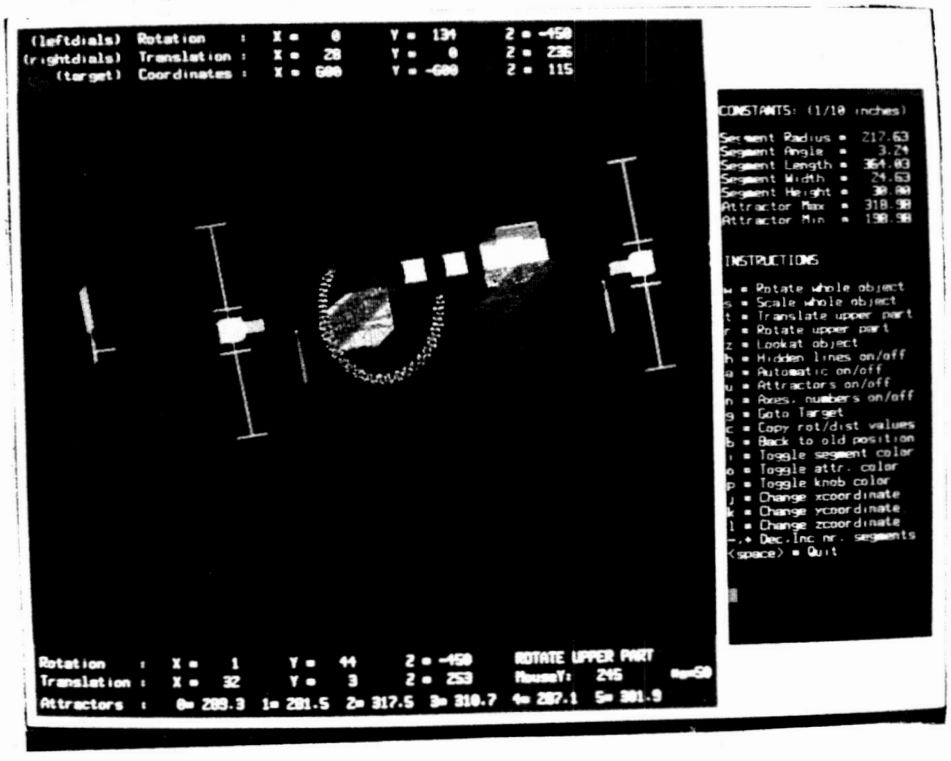


Figure 6. HOST Attached to Space Station

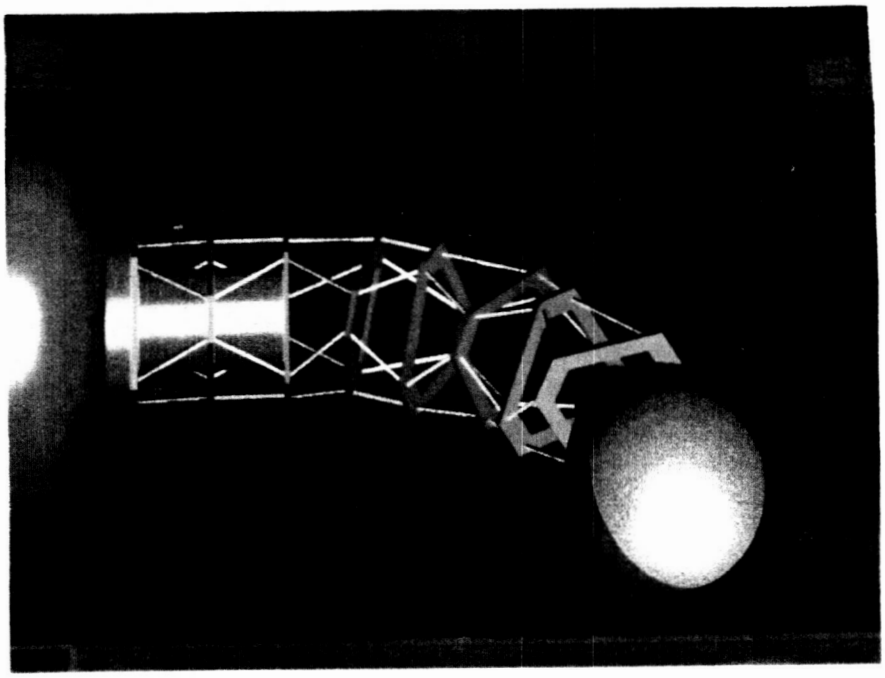


Figure 7. Computer Simulation of POWER

Several areas could use further definition. In particular: 1) a pod design incorporating all requirements and Space Station constraints, 2) loads, dynamic disturbances and further control schemes, 3) orbiter interfaces and initial operations from the orbiter, 4) Space Station storage during early assembly, and 5) a high efficiency actuator design and other items mentioned in an earlier progress report.

In summary, HOST provides many potential advantages as seen on Table II. We recommend that a Phase A definition be completed under direction of a NASA Field Center.

Table II

Advantages of HOST

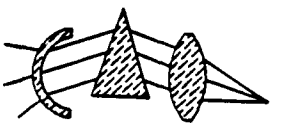
- HOST System Provides Accessibility for Multiple Space Station Tasks without EVA
- HOST Uses Low Risk Technology
- HOST System Saves Crew Time and Reduces Fatique
- POD to Node Attachment Offers Lower Complexity and Direct POD Acess
- Tunnel/Table Structure Alone Offers Attractive Concept for Orbiter Mating with Station by means of Extendable "Jet-Way" from Station to Orbiter
- HOST System Offers Flexibility and Growth
  - Potential Use from Orbiter - Two Modes (POD/HOST or POD only w/RMS)
  - Use as Station Docking Port Extension for Orbiter
  - Fixed Attachment of HOST at Space Station
  - Moveable Attachment Location on Space Station (Rail Mount)
  - Evolution to Space Station Proximity Free Flyer
- System Complements or Substitutes for other EVA/Maneuverable Systems

APPENDIX A

Solid Modeling for the Flexible Robot Arm POWER

Vincent Harrand

July 1987



Center for Applied Optics
University of Alabama in Huntsville (UAH)
Huntsville, Alabama 35899, U.S.A.

This work is performed under the Innovative Research Program for NASA.
NASA-grant number: NAGW-847.

Preface

The project described in this report has been done at the Center for Applied Optics, University of Alabama in Huntsville (UAH), Huntsville, Alabama, U.S.A., in cooperation with the Johnson Research Center of the same university.

Some of the early results of this project have been published in a progress report submitted to NASA [Wessling-87a]. An integral version of the remainder of this report will be published in a final report about the flexible robot arm POWER (Personnel Occupied Woven Envelope Robot) in Fall 1987 [Wessling-87b].

The graphics software is developed on the "IRIS" (Integrated Raster Imaging System) workstation of Silicon Graphics. The IRIS is a high performance workstation and is state-of-the-art at this moment. It performs most of the graphical manipulations in hardware. The heart of the IRIS is a custom VLSI-chip called the "Geometry Engine". The workstation has twelve Geometry Engines (four for matrix multiplications, six for the clipping system, and two for scaling). These Geometry Engines are controlled by the application/graphics processor (the 32-bits Motorola 68020). The video framebuffer has 32 bitplanes and each bitplane has a resolution of 1024 by 768 pixels. The geometry engines in combination with the 32 bitplanes enable the user to do real time animation of a wire model or hardware z-buffering for elimination of hidden surfaces.

I would like to thank Dr. Francis C. Wessling (Associate Director of the "Consortium for Materials Development in Space" and Principle Investigator of the POWER-project) for the fruitful cooperation. I would also like to thank Dr. Amar Choudry (Senior Research Scientist-Center for Applied Optics) for guiding my project and making my stay in the United States possible.

And finally, I would like to thank my Dutch friends in the United States: Paul Janssen for his cooperation in this project (he developed the control algorithms for POWER), Jeroen van der Zijp for his useful comments made on draft versions of this text, and especially Emile Fiesler for his support. He was of great help to me in matters of English language and mathematics.

Vincent Harrand

July 1987

Huntsville, Alabama, U.S.A.

æ

Project Description

The goal of this project is the development of methods and algorithms for solid modeling. The results of this will be used for a NASA-project, which is a feasibility study of a flexible robot arm (POWER: Personnel Occupied Woven Envelope Robot, Grant # NAGW-847 from the Office of Space Science and Applications). This flexible robot arm is designed to be attached to the Space Station. The NASA-project includes a graphical simulation of the flexible robot arm. For this reason and for publicity purposes there is need for a graphical model of the robot arm (including a simple model of the Space Station).

Although the goal of this project is practical, there will be an emphasis on the theoretical aspects of solid modeling. Solid modeling is a very broad subject. Basic concepts, needed for the graphical simulation, such as modeling and viewing transformations, wire models, shaded polygon models, etc. are assumed to be known by the reader. For more information see [Newman-81], [Rogers-76] and [Rogers-85]. The structure of the flexible robot arm is complex. The representation of solids must not constrain the shape of it. Therefore the major research goal is the mathematical description of free-form surfaces and the rendering of these surfaces on the screen.

Abstract

The purpose of this project is solid modeling for the flexible robot arm POWER (Personnel Occupied Woven Envelope Robot).

The first three chapters describe the solid modeling part. The first chapter gives an introduction to solid modeling; what kind of representations of solids do exist and what are their corresponding (dis)advantages. The second chapter deals with the mathematical description of surfaces of solids. The author has chosen the Bezier representation of surfaces. The actual surface is approximated with a recursive subdivision algorithm. In order to perform the subdivision in hardware the original algorithm of [Lane-80a] has been reformulated by the author. The developed algorithm gives, with the standard built-in hardware of the IRIS workstation, a reduction of computing time of more than 60%. Chapter three describes the rendering of the mathematical model on the screen. The shading is determined by a local illumination model. This model provides three aspects of natural light, i.e. diffuse reflection, specular reflection, and ambient light. The visual appearance of the model on the screen can be enhanced by applying Gouraud shading. The intersection of surfaces and the removal of hidden surfaces is solved by making use of the z-buffer algorithm.

And finally, chapter four gives an introduction about the flexible robot arm and describes the application of the theory of the first three chapters for modeling the flexible robot arm.

List of Mathematical Symbols Used in this Text

X $\equiv$ capital letter denotes matrix (contents can be derived from context)
I $\equiv$ identity matrix
X^* $\equiv$ matrix X with the order of the rows reversed
X^{-1} $\equiv$ inverse of matrix X
X^t $\equiv$ transpose of matrix X

s, t, u, v $\equiv$ parametric variables
P_i, P_{ij} $\equiv$ control point of curve or surface (vector consists of x, y, and z)

$\circ$ $\equiv$ innerproduct of two vectors
$[0, 1]$ $\equiv$ closed interval, continuous values
$\binom{m}{n}$ $\equiv$ combinations $(m, n) =: \frac{m!}{n!(m-n)!}$ $m \geq n$ $m, n = 0, 1, \ldots$

Contents

List of Figures

1 Representations of Solids

1.1 Introduction

This section is mainly based on [Requicha-80], [Requicha-81], [Requicha-83], and [Eastman-84]. There exists a substantial amount of techniques for solid modeling, for example Constructive Solid Geometry (CSG), octree/voxels, triangulations, Bezier/B-spline surfaces, and wire frames. The internal representation of an abstract solid in a solid modeler is very important in relation to a specific application. For example every method has constraints on surface complexity and derivation of geometrical properties. None of the known representations can be characterized as the best for every application.

Geometric algorithms do not manipulate physical solids; rather, they manipulate data which represent solids. Suppose that flat-faced solid polyhedra have to be represented and that an edge-based approach is taken. For each edge its begin and end point are stored. It is clear that the edges are represented unambiguously, but that is not what has to be represented. Therefore the following questions are important [Requicha-81]:

- Do edges supply enough information about a solid polyhedron to make it possible to compute the appearance, volume and other geometrical properties of the polyhedron?
- Does an arbitrary collection of edges represent a solid polyhedron? If not, how can one be assured that geometric algorithms operate on valid data.
- How does one determine if two ostensibly different representations correspond to the same solid?
- Can some geometric properties be computed from one representation, but not (or better) from another?
- Which representation is the best?

A general framework for classification of representations is needed for answering these questions. This will be discussed in section 1.2. With this framework it is possible to show the most important characteristics of each representation. In section 1.3 an overview is given of some important representations with their characteristics.

1.2 Mathematical properties of representations of solids

First of all, the properties of an abstract solid have to be defined:

- Rigidity: an abstract solid must have an invariant configuration or shape which is independent of the solid's location or orientation.

- Description of volume: a solid must have an interior, and a solid's boundary can not have isolated or dangling portions.
- Closure under rigid motions and certain Boolean operations: rigid motions (translations and/or rotations) or operations that add or remove material (welding, machining) must, when applied to solids, produce other solids.
- Finite describability: there must be some finite aspect of solids (e.g. a finite number of faces) to ensure that they are representable in computers.

A representation scheme is a relation between abstract solids and representations. The modeling space M consists of all the abstract solids. The collection of all syntactically correct representations is called a representation space R. The representation scheme s is defined as the relation $s : M \rightarrow R$.

- Domain: the domain D of a representation scheme characterizes the descriptive power of the scheme, the domain is the set of entities representable in the scheme. In most cases the domain of the representation scheme will not equal the modeling space M.
- Range: The range V of a representation scheme is the set of representations which are valid. To ensure the representational validity we either have to check this after construction of the database, or representation schemes are needed in which all the representations are valid. Normally the range of the representation scheme is a subset of the representation space R.
- Completeness: A representation is unambiguous or complete if it corresponds to a single object in the domain. A representation scheme s is unambiguous or complete if all of its valid representations are unambiguous (if the inverse relation s^{-1} is a function).
- Uniqueness: a representation r is unique if its corresponding object does not admit other representations in the scheme (if $s(s^{-1}) = r$). A representation scheme is unique if all of its valid representations are unique (if s is a function).

The foregoing definitions may informally be summarized as follows. A representation is invalid if it does not correspond to any solid. A valid representation is ambiguous if it corresponds to several solids. A solid has nonunique representations if it can be represented in several ways by the scheme.

Representation schemes which are both unambiguous and unique are highly desirable because they are one-to-one mappings. This implies that distinct representations in such schemes correspond to distinct objects, and therefore object equality can be determined. Equality assessment in schemes which are

unambiguous but not unique requires more elaborate techniques. For example, two sets of points may be tested for equality by determining whether their symmetric difference is the null set.

Most representation schemes for geometric entities are nonunique for at least two reasons:

- substructures in a representation may be permuted, and
- distinct representations may correspond to differently positioned but congruent copies of a single geometric entity.

In addition to these formal properties it is possible to define some informal properties. These informal properties are important for practical usage, but can not be formalized in a useful way.

- Conciseness: the amount of data that has to be stored for a particular solid must be as low as possible.
- Ease of creation: the ease with which representations may be created by users of modeling systems, especially if the users are human, must be as low as possible.

1.3 Evaluation of different solid modeling techniques

In this section some important representation schemes are discussed. For each scheme the characteristics and the typical applications are given.

Finite-point-sets If a computer representation of a physical solid has to be made, an usual approach consists of measuring a large number of points lying on the boundary of the object. This representation is ambiguous.

Voxels The voxel (volume element) method is essentially a spatial occupancy enumeration scheme. A solid consists of all small cubes that lie in a fixed spatial grid. Spatial arrays are unambiguous and unique (except for positional uniqueness), but they are quite verbose. A typical application is the representation of organic structures. The data is acquired by making cross-sections of a structure (computer tomography). Each cross-section defines one layer of cubes of the voxel-representation. The voxel representation is one of the methods for constructing an unambiguous representation from an ambiguous finite-set-of-points representation.

Constructive Solid Geometry (CSG) CSG uses combinations of solid primitives. These primitives can be combined via certain operations, such as a motion (translation/rotation) and/or a Boolean operation. The primitives are simple solids, for example a cube, sphere, cylinder, and a pyramid. CSG schemes are unambiguous, but not unique. The domain of the CSG

scheme depends on the used primitives and operations. All the representations made with this scheme are valid, concise and easy to create. A typical application of this scheme is the representation of (unsculptured) mechanical parts.

Boundary representation A solid is represented by segmenting its boundary into a finite number of faces, and each face is represented by (for example) its bounding edges and vertices. Faces should satisfy the following conditions:

- A face of an object is a subset of the object's boundary.
- The union of all the faces of an object equals the object's boundary.
- A face must have an area and must not have dangling edges or isolated points.

Planar faces may be represented by their bounding edges, but nonplanar faces have to be represented in a different way. Nonplanar surfaces can be represented by either Bezier or B-spline techniques. In order to get an unambiguous boundary representation scheme, the faces have to be represented unambiguously. The domain of the boundary representation is at least as rich as those of CSG-schemes. [Requicha-80] gives some examples of topological and metric conditions for checking the validity of a boundary representation. Boundary representations are quit verbose and sometimes difficult for humans to construct.

The different representation schemes with their corresponding (dis)advantages gives rise to hybrid solid modelers. These solid modelers use combinations of schemes and combine the power of two schemes. Bidirectional conversion between most of the schemes is not possible. And if the conversions are possible, then they are only known by a very small group of researchers [Requicha-83].

The author has chosen the boundary representation because of the "unlimited" domain of this representation. The domain mainly depends on the chosen representation for sculptured surfaces. In this case the Bezier representation has been chosen for description of surfaces. Some general disadvantages of the scheme (representations are difficult to create and rather verbose) are not important in this project. The Space Station and the flexible robot arm are only defined once by the programmer.

The definition of the planar faces (including the surface normal) is very straightforward and can be found in [Rogers-85]. The same illumination model as for Bezier surfaces can be used for planar surfaces. This illumination model is described in chapter 3.2. In the next chapter the Bezier representation of surfaces will be discussed.

2 Surface Description and Generation

2.1 Introduction

The representation of the surface of various kinds of solids requires some special mathematical methods, because the classical mathematics does not provide adequate methods for conveniently creating surfaces that will satisfy certain design criteria. In this section a brief survey of mathematical constructions used for defining a curve or a surface is given.

For a plane curve, the explicit nonparametric equation takes the general form:

$$y = f(x)$$

In this form, there is only one y value for each x value. This explicit form can not represent closed or multiple-valued curves. This limitation can be overcome by using an implicit equation of the general form:

$$f(x, y) = 0$$

Both explicit and implicit nonparametric equations are axis dependent. And moreover, geometric modeling requires surfaces that are bounded in some sense which can not be represented by a nonparametric function at all. This is one of the most important reasons for using parametric equations. When using a parametric representation, a space curve is given by a set of three functions $x = x(t)$, $y = y(t)$, and $z = z(t)$ of a parameter t. A surface is represented by $x = x(u, v)$, $y = y(u, v)$, and $z = z(u, v)$. Normally the parametric variables are bounded on the interval [0,1]. This restriction gives rise to curve and surface boundaries. In practice, the parametric representation is still not suitable for geometric modeling, because each shape has its own equations and therefore needs its own computer programs. It is very convenient to have one general form for describing an arbitrary curve or a surface. Such general forms exist and one of them is the Bezier form.

2.2 Bezier surfaces

There exists a lot of good reference material about Bezier curves and surfaces, for example [Rogers-76] and [Mortenson-85].

The Bezier curves and surfaces are named after P. Bezier. He worked at the French automobile company of Renault, where he developed the Unisurf system for designing sculptured surfaces of automobile bodies.

Bezier started with the principle that any point on a curve segment must be given by a parametric function of the following form:

$$B(t) = \sum_{i=0}^{m} P_i f_i(t) \qquad t \in [0, 1] \tag{1}$$

where the vectors P_i represent $m + 1$ vertices of a polygon. These vertices are called control points. The blending function $f_i(t)$ determines the effects or contributions of the control points to the resulting curve. This means that the control points only determine the shape of the curve and that the control points, except for the begin and end points, lie not on the curve itself. Bezier wanted to establish certain properties and was looking for specific blending functions to meet these requirements. Some of these properties are:

1) The functions must interpolate the first and last control points; that is, the curve segment must start at P_0 and end at P_m.

2) The tangent at P_0 must be given by $P_1 - P_0$, and the tangent at P_m by $P_m - P_{m-1}$. This gives direct control of the tangent to the curve at each end. This is important for combining two or more curves.

3) The functions $f_i(t)$ must be symmetric with respect to t and $(1-t)$. This means that the sequence of the vertex points can be reversed without changing the shape.

Some more general properties of Bezier curves are [Newman-81]:

4) The parametric formulation of the Bezier curve allows it to represent multiple valued curves.

5) A Bezier curve is independent from the chosen coordinate system used to measure the control points.

6) Bezier curves are variation diminishing. This means that the curves are very smooth and that there are no oscillations or other irregularities.

Bezier chose a family of functions called Bernstein polynomials to satisfy these conditions:

$$J_{m,i}(t) = \binom{m}{i} t^i (1-t)^{m-i} \tag{2}$$

So equation 1 becomes:

$$B(t) = \sum_{i=0}^{m} P_i J_{m,i}(t) \qquad t \in [0,1] \tag{3}$$

For most of the applications a cubic curve is used, i.e. there are four control points and $m = 3$. Figure 1 depicts the blending function curves for $m = 3$. The first control point P_0, whose contribution to the curve's shape is determined by

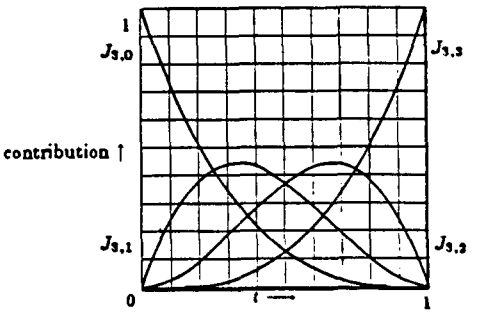


Figure 1: blending functions

$J_{m,0}(t)$ is the most influential when $t = 0$. The other control points do not contribute to $B(t)$ for $t = 0$, since their associated blending functions are zero. A symmetrical situation occurs for P_3 when $t = 1$. Control points P_1 and P_2 are most influential when $t = \frac{1}{3}$ and $t = \frac{2}{3}$, respectively. So there is a shift in the influence of each control point as the parametric variable t moves through its range from 0 to 1 (this is called blending).

When joining two Bezier curves, the first (and second) derivatives at begin and end points have to be evaluated. They are (for $m = 3$):

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

$$B'(t) = -3(1-t)^2 P_0 + (9t^2 - 12t + 3)P_1 + (-9t^2 + 6t)P_2 + 3t^2 P_3$$

$$B'(0) = 3(P_1 - P_0)$$
$$B'(1) = 3(P_3 - P_2)$$

$$B''(t) = (-6t + 6)P_0 + (18t - 12)P_1 + (-18t + 6)P_2 + 6tP_3$$

$$B''(0) = 6(P_0 - 2P_1 + P_2)$$
$$B''(1) = 6(P_3 - 2P_2 + P_1)$$

This illustrates that the first derivative of the Bezier curve at the begin and end points depends only on the nearest two control points. And the second derivative depends on the nearest three control points. Continuity requirements between adjacent Bezier curves can therefore be easily met.

This model can be extended for defining surfaces. This can be done by taking the cartesian product of the basis function (Bernstein) with respect to

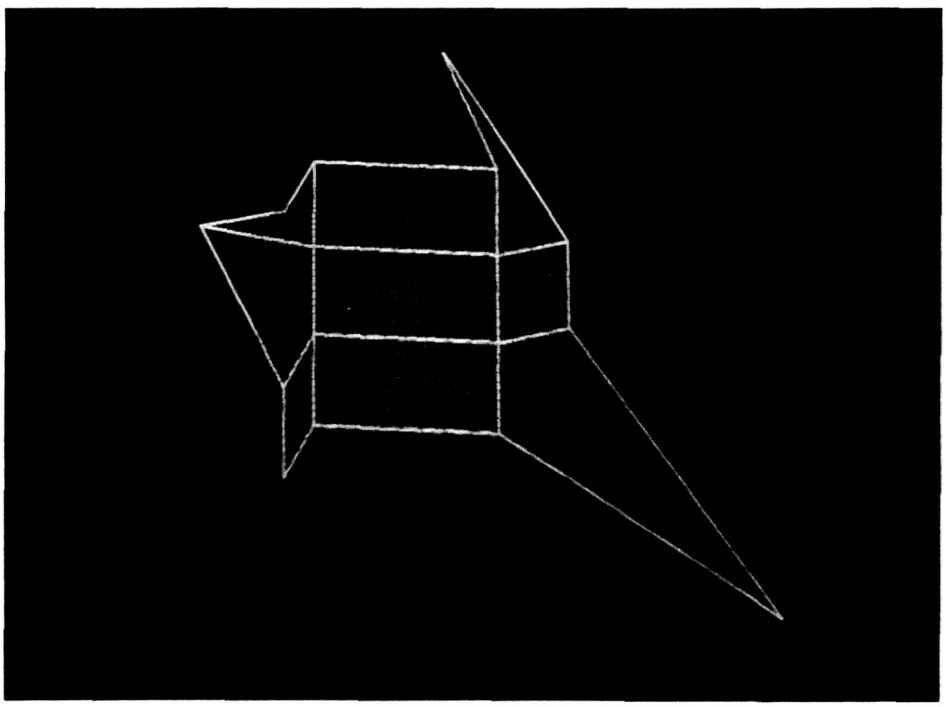


Figure 2: Bezier surface with mesh of 16 control points

the two orthogonal directions. This results in the following equation:

$$B(u,v) = \sum_{i=0}^{m} \binom{m}{i} u^i (1-u)^{m-i} \left(\sum_{j=0}^{n} \binom{n}{j} v^j (1-v)^{n-j} \right) P_{ij} \qquad (4)$$

Equation 4 can be simplified to:

$$B(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} J_{m,i}(u) J_{n,j}(v) P_{ij} \qquad (5)$$

With $(u,v) \in [0,1] \times [0,1]$, m and n are the degrees of the polynomial with respect to the u and v directions, and the P_{ij} are the control points. Figure 2 depicts a mesh of 16 control points $(m,n = 3)$ with the corresponding Bezier surface.

2.3 Approximation of the surface with recursive subdivision

This section describes the approximation of the Bezier surface, by means of a recursive subdivision of the convex hull formed by the control points. An introduction to this subject is given by [Lane-80b]. A more detailed description can be found in [Lane-80a].

If the Bezier representation of a surface is known, it can be displayed on a screen with curves of constant parameter. See figure 2. This is very straightforward and gives no special problems. In order to display the surface with shading techniques, in each point the normal vector of the surface has to be known. The calculation of the surface normal is computationally very expensive [Mortenson-85]. Therefore, the surface normal will not be calculated directly, but by means of an approximation of the surface. The approximation of the surface will be acquired by a recursive subdivision of the surface, i.e. the polynomial over the parametric range $[0, 1]$ is replaced by a combination of two polynomials, one over $[0, \frac{1}{2}]$, the other over $[\frac{1}{2}, 1]$. Each of them is reparametrized so that it is defined over $[0, 1]$. As the process of subdivision continues, the polyhedron formed by the control points approaches the actual surface. In rendering, the subdivision process is carried out until the convex hull of the control points is planar with linear edges. The calculation of the surface normals then becomes very easy and is described in section 2.5.

2.3.1 Proof of the subdivision method

A short outline of the mathematical proof of the subdivision algorithm in [Lane-80a], with some additional remarks, is given here.

The following properties of the Bernstein polynomials will be important in the formulation of the subdivision algorithm.

Lemma 2.1 *(Convex Hull Property) The Bernstein basis function is nonnegative on* $[0, 1]$ *and sums identically to 1, i.e.*

$$\sum_{i=0}^{m} J_{m,i}(t) = \sum_{i=0}^{m} \binom{m}{i} t^i (1-t)^{m-i} \equiv 1 \tag{6}$$

where $J_{m,i}(t) \geq 0, \forall t \in [0, 1]$, *where* $i \in (0, 1, \ldots, m)$ *and* m *is a positive integer.*

Proof: The basis function is nonnegative on $[0, 1]$. *The binomial expansion theorem is:*

$$\sum_{i=0}^{n} \binom{n}{i} a^{n-i} b^i = (a+b)^n$$

This in combination with equation 6 gives:

$$\sum_{i=0}^{m} \binom{m}{i} t^i (1-t)^{m-i} = (t+1-t)^m = 1$$

In fact the Bernstein basis function is a weighted average of the control points (blending). And this average will always fall within the convex hull of the control points.

□

Lemma 2.2 $J_{m,i}(t) = J_{m,m-i}(1-t)$
Proof:

$$J_{m,i}(t) = \binom{m}{i} t^i (1-t)^{m-i} = \binom{m}{m-i} t^i (1-t)^{m-i} \Rightarrow$$

$$J_{m,i}(t) = \binom{m}{m-i} (1-t)^{m-i} t^{m-(m-i)} = J_{m,m-i}(1-t)$$

The Bernstein basis function is symmetrical on the interval $[0,1]$ *with respect to* t *and* $(1-t)$. *This means for example that the sequence of control points can be reversed without changing the shape of the curve.*

□

Theorem 2.1 *(Subdivision theorem)*
Let $B_m[P : 0,1] = B_m[P_0, P_1, \ldots, P_m : 0,1]$ *be defined as the Bernstein polynomial of degree* m *to the polygon* P *on interval* $[0,1]$. *Polygon* P *consists of the control points* $P_0, \ldots, P_m$. *The original curve over the parametric range* $[0,1]$ *is replaced by two curves over the parametric range* $[0, \frac{1}{2}]$ *and* $[\frac{1}{2}, 1]$ *respectively. Then the curves are reparametrized so that both get defined over the parametric range* $[0,1]$.
Then the reparametrized first half is defined by:

$$B_m[P : 0,1] = B_m[P_0^0, \ldots, P_m^m : 0, \tfrac{1}{2}] \tag{7}$$

and the second half by:

$$B_m[P : 0,1] = B_m[P_m^m, P_m^{m-1}, \ldots, P_m^0 : \tfrac{1}{2}, 1] \tag{8}$$

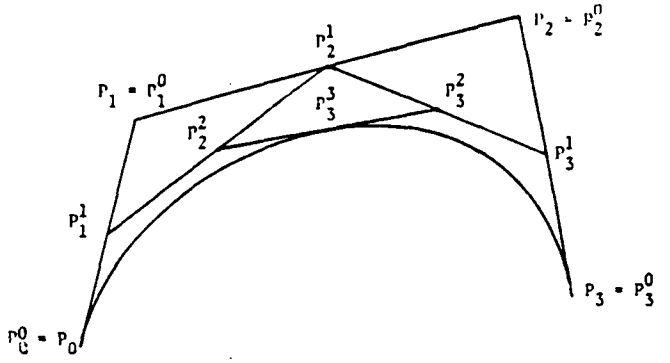


Figure 3: subdivision of a cubic Bernstein-based polynomial [Lane-80a]

where (see figure 3):

$$P_i^k = \begin{cases} (P_{i-1}^{k-1} + P_i^{k-1})/2 & k = 1, \ldots, m \\ P_i & k = 0 \end{cases} \tag{9}$$

The first equality (7) can be proven by induction on m (see [Lane-80a]). The second equality (8) can then be proven by lemma 2.2, because the symmetric relationship must hold.

□

If this splitting construction is applied to each of the polygons $[P_0^0, P_1^1, \ldots, P_m^m]$ and $[P_m^m, P_m^{m-1}, \ldots, P_m^0]$, four polygons are generated, which, when concatenated, form a polygon ψ_m^2 of $4m+1$ vertices. Defining $\psi_m^k[P]$ as the polygon derived after k iterations of this algorithm (ψ_m^k would have $2^k m + 1$ vertices), it has to be proven that:

$$\lim_{k \to \infty} \psi_m^k[P] = B_m[P : 0, 1] \tag{10}$$

Theorem 2.2 *(Convergence)*

Let $B_m[P : 0, 1]$ be the vector-valued Bernstein approximation of degree m to the polygon $P = [P_0, \ldots, P_m]$, and define the polygon $\psi_m^k[P]$ as above. Then equation 10 must hold.

The proof of this algorithm can be found in [Lane-80a]. The proof is based on lemma 2.1. The polyhedron formed by the convex hulls of all the subcurves will get closer and closer to the actual curve, because the area of each convex hull is getting smaller and each subcurve must lie within the corresponding convex hull. So the whole subdivision process converges to the actual Bezier curve.

2.3.2 The subdivision algorithm

Based on theorem 2.1 an algorithm for subdivision can be derived. Given the polynomial coefficients $P \equiv [P_0, P_1, \ldots, P_n]$ in terms of the Bernstein basis on $[0,1]$ compute the subcurves $Q \equiv [Q_0, Q_1, \ldots, Q_n]$ and $R \equiv [R_0, R_1, \ldots, R_n]$.

Procedure Curvesplit (P, Q, R, n)

step 1: [initialize]
$Q \leftarrow P$
$R_n \leftarrow Q_n$
step 2: [compute coefficients]
for $j = 1$ **to** n
begin
$QTMP2 \leftarrow Q_{j-1}$
for $k = j$ **to** n
begin
$QTMP1 \leftarrow QTMP2$
$QTMP2 \leftarrow (Q_{k-1} + Q_k)/2$
$Q_{k-1} \leftarrow QTMP1$
end
$Q_n \leftarrow QTMP2$
$R_{n-j} \leftarrow QTMP2$
end
return

A similar algorithm exists for surface subdivision. The surface $B(u, v)$ is subdivided in two steps, namely:

1) Subdivision of the surface in the u direction, which results in two subsurfaces.

2) Subdivision of these two subsurfaces in the v direction, which results in a total of four subsurfaces.

The surface subdivision algorithm Surf_Split(P, Q, R, S, T, m, n) can be found in Appendix B.

In fact these algorithms do only one subdivision. A good approximation of the surface can be obtained by a recursive subdivision, in the following way:

```
Procedure Surface_Generation (P, m, n)
  if 'planarity of P' ≤ Tolerance then
        Draw_on_Screen(P)
  else
     begin
        Surf_Split(P, Q, R, S, T, m, n)
        Surface_Generation(Q, m, n)
        Surface_Generation(R, m, n)
        Surface_Generation(S, m, n)
        Surface_Generation(T, m, n)
     end
  return
```

Notes on the algorithm:

(a) recursion depth

The recursive subdivision of the surface is a logarithmic process. After k iterations there are 4^k subsurfaces. The screen has approximately 1M pixels. If the surface is subdivided until pixel-size, the maximum recursion depth is 10. This limits the number of iterations and hence the amount of dynamic data-storage that is required.

(b) cracks

If the subpatches are relatively big, it is possible that there are some cracks (small empty areas between adjoining patches) in the total surface, because the subpatches are not completely planar, nor are their edges straight. In [Tamminen-85] a "Surface Integrity Filter" is described. This filter is based on quadtrees. This filter can be avoided by requiring that the planarity of P is within a tolerance of less than 1 pixel.

(c) planarity estimation of surface P

If the planarity of surface P is within a certain tolerance (for example 1 pixel) then the subdivision is ready and the patch P can be displayed on the screen. [Lane-80a] and [Mudur-86] describe some algorithms for planarity estimations. These estimations are based on an approximation of the planarity of the convex hull rather than on the surface itself (see lemma 2.1). These estimations involve a lot of computation. In practice the area of a patch will be chosen very small (towards one or a small number of pixels). This is important for the visual appearance of the surface on the screen (smoothness and no cracks). Therefore (in this implementation) not the planarity but the area of the patch is estimated. And an estimation of the area is very simple to acquire.

(d) The procedure Draw_on_Screen(P) is a formalization of the theory discussed in chapter 3 "The Rendering of the Mathematical Model".

2.3.3 Reformulation of the subdivision algorithm

Each patch can have hundreds (or even thousands) of subpatches. Every reduction in computation time of the subdivision algorithm is therefore important.

The Bezier algorithm can be described as a series of matrix multiplications. A matrix multiplication can be done in (dedicated) hardware. If the matrix-approach can be extended for the subdivision algorithm, there will be a tremendous reduction of computation time, because implementation in hardware is much faster then in software. In this project, the hardware of the IRIS workstation can be used (the IRIS hardware performs a 4 by 4 matrix multiplication intended for viewing transformations, etc.; this facility is user-accessible).

Derivation of the matrix formulation of Bezier curves:

$$B(t) = \sum_{i=0}^{m} J_{m,i}(t) P_i \tag{11}$$

where

$$J_{m,i}(t) = \binom{m}{i} t^i (1-t)^{n-i} \tag{12}$$

for $m = 3$ equation 12 becomes:

$$\begin{aligned} J_{3,0} &= (1-t)^3 = -t^3 + 3t^2 - 3t + 1 \\ J_{3,1} &= 3t(1-t)^2 = 3t^3 - 6t^2 + 3t \\ J_{3,2} &= 3t^2(1-t) = -3t^3 + 3t^2 \\ J_{3,3} &= t^3 \end{aligned}$$

The Bezier curve in matrix notation:

$$B(t) = \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} \tag{13}$$

If the three matrices on the right-hand side of equation 13 are represented by respectively T, J, and P, then:

$$B(t) = TJP \tag{14}$$

Subdivision is a technique for deriving the set of four control points P_i' defining a curve $B(u)$ such that $B(u) = B(t)$, where the u-range is the first half of the t-range, i.e.:

(1) $B(u)$ is the first half of the original curve $B(t)$, and

(2) the relation between the u-values and the t-values is: $t = \frac{1}{2}u$.

The curve $B(u)$ is defined as:

$$B(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} JP' = UJP' \tag{15}$$

In order to find P', the equations 14 and 15 are written as a function of one parameter. The first half can be described (with parameter u) as:

$$B(u) = UJP' = \begin{pmatrix} (\frac{1}{2}u)^3 & (\frac{1}{2}u)^2 & \frac{1}{2}u & 1 \end{pmatrix} JP$$

The matrix $\begin{pmatrix} (\frac{1}{2}u)^3 & (\frac{1}{2}u)^2 & \frac{1}{2}u & 1 \end{pmatrix}$ can be rewritten as:

$$\begin{pmatrix} \frac{1}{8}u^3 & \frac{1}{4}u^2 & \frac{1}{2}u & 1 \end{pmatrix} = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{8} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = US$$

$B(u) = UJP' = USJP \Rightarrow$
$P' = J^{-1}SJP$

The first three matrices on the right-hand side of the equal sign can be precalculated (this will be represented by S') and subsequently used to do a curve subdivision at the cost of a single matrix multiplication. The calculation of S' can be found in Appendix C.

S' and P' can be calculated as follows:

$S' = J^{-1}SJ$
$P' = S'P$

The matrix S' is only suitable for the first half of the curve. In order to get the second half of the curve $B(t)$ (i.e. the control points P''), it is possible to:

a) reverse the order of the rows[1] of P and S' (this can be done because of lemma 2.2), or

b) redefine the matrix S' (this will result in the matrix S'').

The last option is computationally the best, therefore S'' has to be derived.

a) and b) can be more formally described as:

a) $P'' = S'^* P^*$

b) $P'' = S'' P$

$S'' P = S'^* P^*$

Lemma 2.3 $P^* = I^* P$ $\qquad I^* = m$ *by* m, $P = m$ *by* n
The order of the rows of P *can be inverted by multiplying* I^* *with* P.
The general definition of a matrix multiplication is:

$$c_{ij} = \sum_{h=1}^{m} I^*_{ih} P_{hj} \qquad i = 1, \ldots, m \quad j = 1, \ldots, n \tag{16}$$

$$I^*_{ih} = 0 \qquad \text{iff} \quad i + h \neq m + 1$$
$$I^*_{ih} = 1 \qquad \text{iff} \quad i + h = m + 1 \Rightarrow h = m + 1 - i$$

Equation 16 can now be simplified to:

$$c_{ij} = P_{m+1-i,j} = P^*$$

□

With lemma 2.3 P^* can be replaced by $I^* P$, so:

$S'' P = S'^* I^* P \Rightarrow$
$S'' = S'^* I^*$

The calculation of S'' can be found in Appendix C.

[1]notation: X^* means that the order of the rows of matrix X has been inverted, thus $X^*_{ij} := X_{m-i+1,j}$, where X has m rows.

With these two matrices (S' and S'') it is possible to divide the curve in two separate curves. For surface subdivision this model has to be extended. The subdivision of a Bezier surface patch results in 4 subpatches. This can be done by subdividing each column and subsequently subdividing each row. The column subdivision results in two subpatches. Each of these subpatches is divided in another direction (row-subdivision), which results in 4 subpatches. S' and S'' can be used for the column subdivision. For the row subdivision two other matrices have to be defined.

There are two methods for doing the row subdivision, namely:

a) transpose the matrix P, and use S' to obtain the first subpatch, or

b) define a new matrix S'''.

This can be more formally described in the following way:

a) $P'''^t = S'P^t$

b) $P''' = PS'''$ (note the reversion of the matrix order)

a) and b) combined:

$$S'P^t = (PS''')^t \Leftrightarrow$$
$$S'P^t = S'''^tP^t \Rightarrow$$
$$S' = S'''^t \Leftrightarrow S''' = S'^t$$

The calculation of S''' can be found in Appendix C.

The second subpatch for row subdivision can be obtained by either:

a) reverse the columns of S''' and P, or

b) define a new matrix S''''

Lemma 2.4 *The order of the columns of a square matrix A (n by n) can be reversed by multiplying with I^*. The proof is analogue to the proof of lemma 2.3*

□

a) and b) can be more formally described with lemma 2.4:

a) $P'''' = PI^*S'''I^*$

b) $P'''' = PS''''$

$$PS'''' = PI^*S'''I^* \Rightarrow$$
$$S'''' = I^*S'''I^*$$

With lemma 2.3 the equation for S'''' can be simplified to:

$$S'''' = S'''^*I^*$$

The calculation of S'''' can be found in Appendix C.

There are four matrices derived for subdivision of surfaces. The new algorithm for subdivision of surface P is as follows:

$P \equiv P_{ij}$ is the original surface. $Q \equiv Q_{ij}$, $R \equiv R_{ij}$, $S \equiv S_{ij}$, and $T \equiv T_{ij}$ are the four resulting subsurfaces (with $i, j = 1, \ldots, 4$).

Procedure Surfsplit (P, Q, R, S, T)

begin
[split P in u direction]
$U = S'P$
$B = S''P$
[split U in v direction]
$Q = US'''$
$S = US''''$
[split B in v direction]
$R = BS'''$
$T = BS''''$
end

Evaluation

The new algorithm is very elegant compared to the solution of [Lane-80a] (see appendix A). Both algorithms were implemented for comparing their performances. The new algorithm gives more than 60% reduction in computing time. The IRIS workstation provides only a software driven 32 bits (hardware) matrix multiplier. However, if special hardware could be used, a further reduction in computing time would be realized.

In [Pulleyblank-87] the feasibility of a VLSI-chip for ray-tracing bicubic patches is studied. One part of the design is the subdivision of the surface.

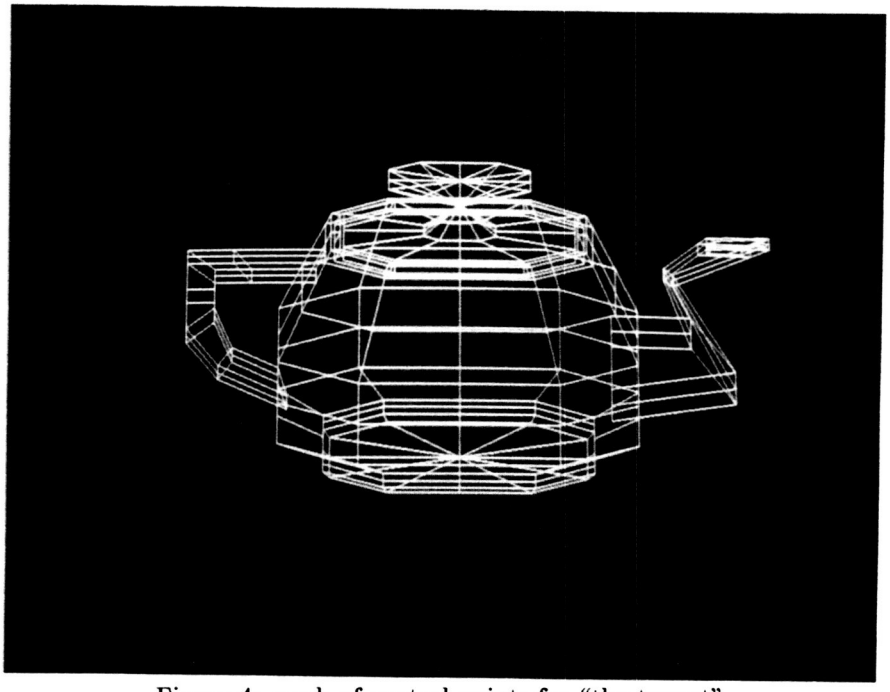


Figure 4: mesh of control points for "the teapot"

They use a hardware version of Lane's algorithm. The new matrix algorithm would give an improvement for two reasons:

- it requires less hardware than a direct implementation of Lane's algorithm, and moreover
- the matrix multiplier can also be used for the viewing and modeling transformations (projections, translations, and rotations).

The subdivision algorithm, based on matrix multiplications, is a major step towards a 'real time solid modeler'. (At this moment, real time solid modeling is an important research goal for companies like Silicon Graphics [Robertson-87: Pushing the Limits of 3D technology].)

2.4 Composite surfaces & special cases

In practice one can not model complex objects with only one Bezier surface patch. In order to accomplish complex surfaces, it is necessary to join several surface patches to form a composite surface. It is important that the composite surface has no discontinuities (C^0-continuity) and that the first derivative of the surface has no discontinuities either (C^1-continuity). These conditions can be

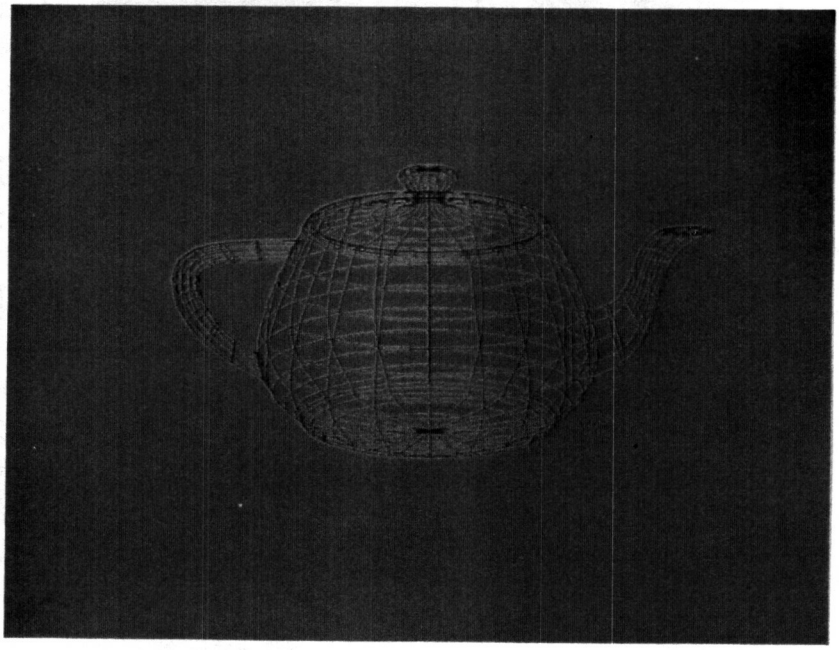


Figure 5: surface of "the teapot" with curves of constant parameter

easily met by controlling the position of the control points. In order to achieve C^0-continuity the boundary curves (of the shared boundary) must coincide.

The four corner control points lie on the patch. The other (8) boundary points control the slope of the boundary curves (the vector from a corner point to the nearest boundary point is called a tangent vector). The four inner control points define the cross slope along the boundary curves (the so-called twist vectors). In order to get C^1-continuity the corresponding vectors of both surfaces must be collinear.

Figure 4 shows the mesh of control points of "the teapot". See for more information about "the teapot" [Crow-87]. "The teapot" consists of 28 Bezier surface patches. Figure 5 depicts the surface of "the teapot" with curves of constant parameter. The shaded model of "the teapot" is shown in figure 11.

An other way of combining surfaces is sweeping a surface along a certain curve. Figure 6 and 8 show two examples. Each segment of the tube consists of two Bezier surface patches (front & backside). The definition of the sweeping curve is done by rotating a certain angle around one of more axis. Later on, this model can be extended for sweeping of surfaces along an arbitrary Bezier curve.

Normally a Bezier surface patch has a more or less rectangular grid of control points. In some cases it is necessary to deviate from this principle. For example,

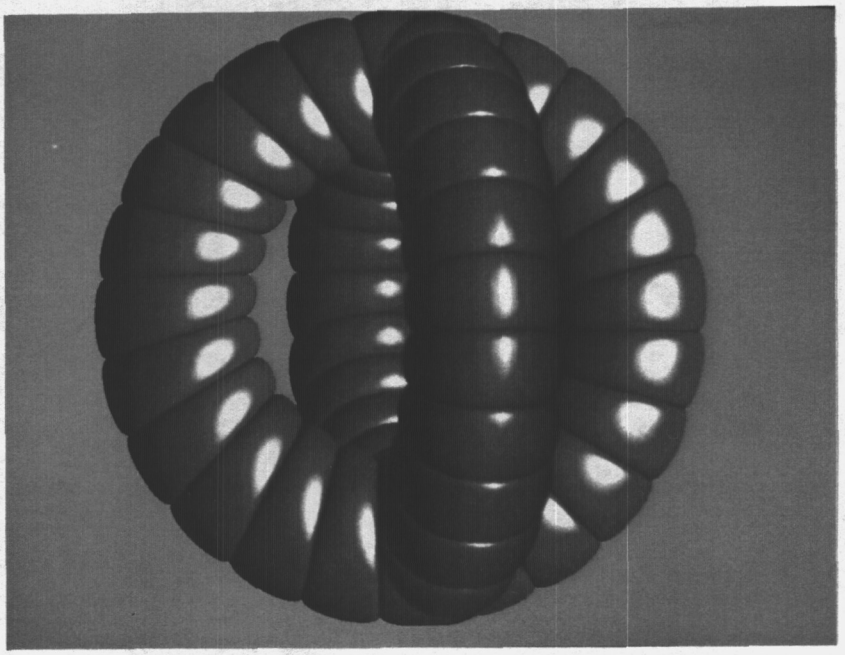


Figure 6: sweeping of surfaces along a curve

for modeling a sphere it is convenient to make use of triangular patches. The upper part of a sphere can be modeled by combining four triangular surface patches. The four coincident vertices form the 'pole' and the four opposite sides the 'equator' of the sphere. A triangular surface can be formed by combining the four control points of one side of the patch. This is called a degenerate surface. Other types of degenerate surfaces are possible, but most of them are of little practical value.

2.5 Determination of surface normals

The purpose of the subdivision algorithm is to approximate the surface and to determinate the surface normals, necessary for the rendering (in particular shading) of the mathematical model on the screen (see chapter 3).

In general the surface normals have to meet certain requirements, namely:

- the normal is outwards (out of the surface) directed,
- the normal has to be normalized, and
- (for Gouraud shading) there has to be a normal in each control point.

If the convex hull formed by the control points of the patch is planar, within a given tolerance, the patch may be treated as linear.

The determination of the surface normals can be done in three ways (with increasing exactness and complexity), namely:

(1) The convex hull is considered to be a planar four sided polygon. In this case the surface normal can be calculated by taking the cross product of two sides (the sides considered to be a vector). So the whole patch has one surface normal, and hence one shade.

(2) The patch is described by a mesh of 16 control points. In fact, the convex hull is a polyhedron and consists of 9 subpatches (see figure 2). These subpatches are considered to be planar. With two edges of each subpatch, the surface normal of the entire subpatch can be calculated by taking the cross product. Therefore, the subpatch gets one shade and the whole patch get nine shades.

(3) Each control point has one, two or four coincident subpatches for a corner, an edge or an inner control point respectively. The normal in each control point can be calculated by taking the average over all the surface normals of the coincident subpatches. With this normal the shade of that point can be calculated (see section 3.2). The shades of the other surface points will be calculated by linear interpolation of the shades at the control points. (see Gouraud shading, section 3.3). This approach will give the best results, because each point on the surface gets its own shade.

3 Rendering of the Mathematical Model

3.1 Introduction

If the mathematical model of an object is known, including the surface normals at certain points on its surface, the object can be displayed on the screen. The first step is the determination of the shade at each point of the surface where the surface normals are known. In this case a local illumination model (with highlighting) is used.

The second (optional) step is a linear interpolation of these shades (at certain points on the surface) throughout the whole surface. This is known as Gouraud shading. If Gouraud shading is not used, the whole (sub)patch gets one shade (see section 2.5).

The last step is the removal of (parts of) surfaces, which are not visible from the point of view (hidden surface elimination). In addition to surfaces, which are not visible, some surfaces intersect with each other. Both problems can be solved with the z-buffer algorithm.

3.2 Illumination model and Shading

A lot of articles are written about shading and illumination models. Some references are [Amanatides-87], [Lorig-86] and [Rogers-85]. An illumination model involves physical and psychological aspects, neither of these aspects will be discussed here.

When a light ray falls on a surface, it can be absorbed or reflected. The amount of reflected light from the surface of an object depends on the direction (and type) of the light source, the surface orientation and the surface properties of the object. This means that the shading calculations are only based on local properties, i.e. the overall setting of the surface in the total scene is ignored (local illumination model).

Illumination models usually consist of a number of components, each component designed to simulate some aspect of light. The used illumination model consists of three components, namely ambient light, diffuse reflection and specular reflection.

Ambient light

Ambient light is a light source, which illuminates all points of the object equally. This light source represents the light that is scattered back from the surroundings (walls). The reflected light is radiated uniformly in all directions. The intensity I can be obtained by:

$$I = I_a k_a \tag{17}$$

where:

I_a = ambient light intensity
k_a = ambient reflection coefficient

In practice $I_a k_a$ is just a constant (the objects in figure 9 have $I_a k_a = 0.4$).

Diffuse reflection

The diffuse reflection term represents light which is emitted from a specific light source. This light strikes the surface and is then uniformly reflected in all directions, therefore the position of the viewer is unimportant. The amount of light which is diffusely reflected depends on the angle between the direction of the point light source and the surface normal. Objects rendered with only diffuse lighting appear as if made of a dull smooth plastic (see figure 8). It is possible that there are more light sources in the scene and all these contributions can be added up. So, the intensity I is:

$$I = \sum_{i=1}^{\#\text{ of lights}} I_{l_i}(k_d(L_i \circ N)) \tag{18}$$

where:

L_i = normalized i-th light vector
N = normalized surface normal
I_{l_i} = light intensity of the i-th light source
k_d = diffuse reflection coefficient

(The objects of figure 9 have '# of lights' = 1, $I_{l_1} = 0.7$, and $k_d = 0.6$)

Specular reflection

The specular part of the model is particularly good for highlights on glossy surfaces. The amount of incident light which is specularly reflected depends on the angle between the reflected light (the angle of reflection is equal to the angle of incidence) and the eye vector (see figure 7). The intensity I can be calculated by:

$$I = \sum_{i=1}^{\#\text{ of lights}} I_{l_i}(k_s(R_i \circ E)^n) \tag{19}$$

where:

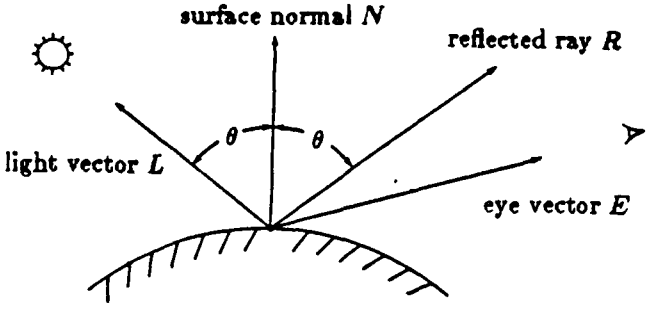


Figure 7: specular reflection

k_s = specular reflection coefficient
R = normalized reflection vector (of light vector)
E = normalized eye vector (position of viewer)

(The objects of figure 9 have $k_s = 0.6$)

The power n denotes how reflective the surface is. By increasing value of n, the highlights become smaller and more sharply defined. A value of infinity means that the surface is a perfect mirror (i.e. light is reflected only in the direction for which angles of incidence and reflection are equal). Figure 8 demonstrates the impact of this parameter.

The reflected light vector R can be calculated with the following formula (see [Lorig-86]):

$$R = 2(L \circ N)N - L$$

Given the surface normal (N), point of view vector (E), and the position(s) of the light source(s) (L_i), the light intensity can be calculated by combining formulas 17, 18, and 19. This results in:

$$I = I_a k_a + \sum_{i=1}^{\#\text{ of lights}} I_{l_i}(k_d(L_i \circ N) + k_s(R_i \circ E)^n) \qquad (20)$$

Some notes:

- If one of the contributions to the light intensity is less then zero, the particular contribution is discarded (made equal to zero).

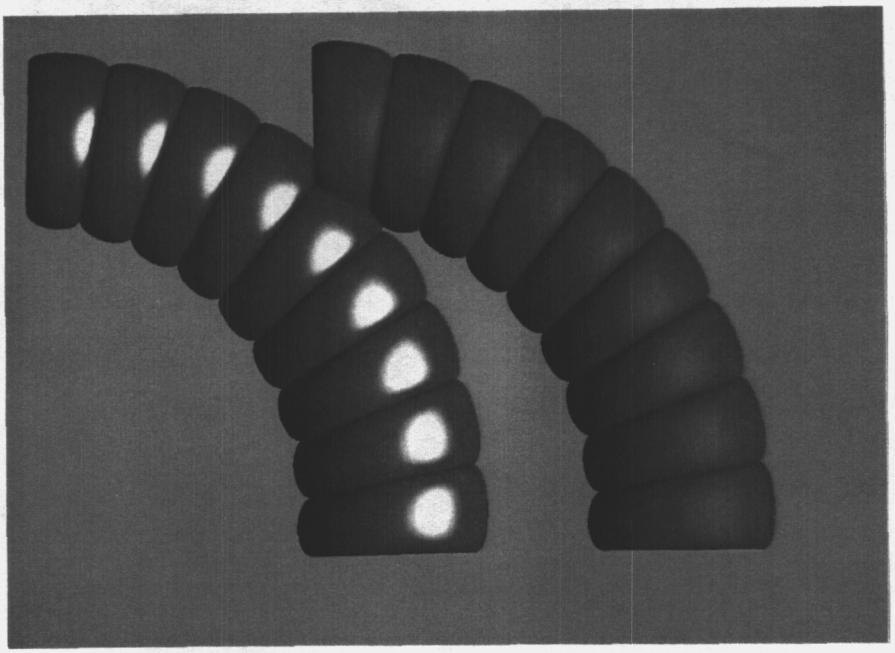


Figure 8: example of a glossy and a matte surface

- The formula results in an intensity value between 0.4 and 1.24 (with the chosen parameter values).
- The objects (figure 9) have 128 shades of one particular color ("0 = dark" and "128 = light" shade of that color). So the intensity value I is a direct mapping to the color lookup-table[2] (with an offset for each color). For example the color lookup-table looks like this (see figure 9):

  $128 - 255 \rightarrow$ shades of red

  $256 - 383 \rightarrow$ shades of gray

  $384 - 511 \rightarrow$ shades of green

  $512 - 639 \rightarrow$ shades of blue

  $640 - 767 \rightarrow$ shades of yellow

  $768 - 895 \rightarrow$ shades of magenta

  $896 - 1023 \rightarrow$ shades of cyan

  The IRIS workstation can handle up to 32 colors (each with 128 shades) simultaneously with the hardware z-buffering.

[2]The contents of the video memory is only a pointer to a table (color lookup-table). This table contains the actual colors. This has two reasons: (a) reduction of video memory and (b) efficient manipulation with colors.

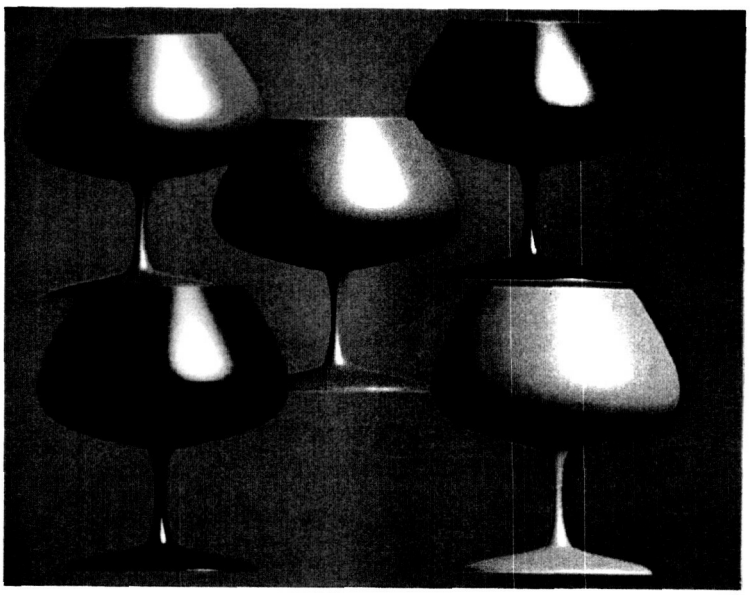


Figure 9: shaded model of wine glasses

- The color of the light source determines the color of the specular component. In this model a light source with white light is assumed. Therefore in the shade range of 80 to 128 (contribution of specular reflection), an increasing white component is added.

3.3 Gouraud shading

If the illumination model is applied to a subdivided Bezier surface using one surface normal for the whole subpatch, a surface with small facets results. A smoother appearance can be obtained using a technique developed by Gouraud [Gouraud-79]. Figure 10 shows two surfaces, one with and the other without Gouraud shading ("The Doughnut Data" by courtesy of [Blinn-87]).

The algorithm is simple. The polygon and the surface normal at each of its vertices are known (see section 2.5). With the surface normal one can calculate the shade in each vertex of the polygon (see section 3.2). The shades of the pixels inside the polygon can be found by linear interpolation of the shades at the vertices. The shades for the points along the edges of the polygon are determined by interpolating linearly between the shades at the vertices. The shades for all of the interior points of the polygon are determined by interpolating linearly between the pairs of edge points that lie along each scan line. The IRIS

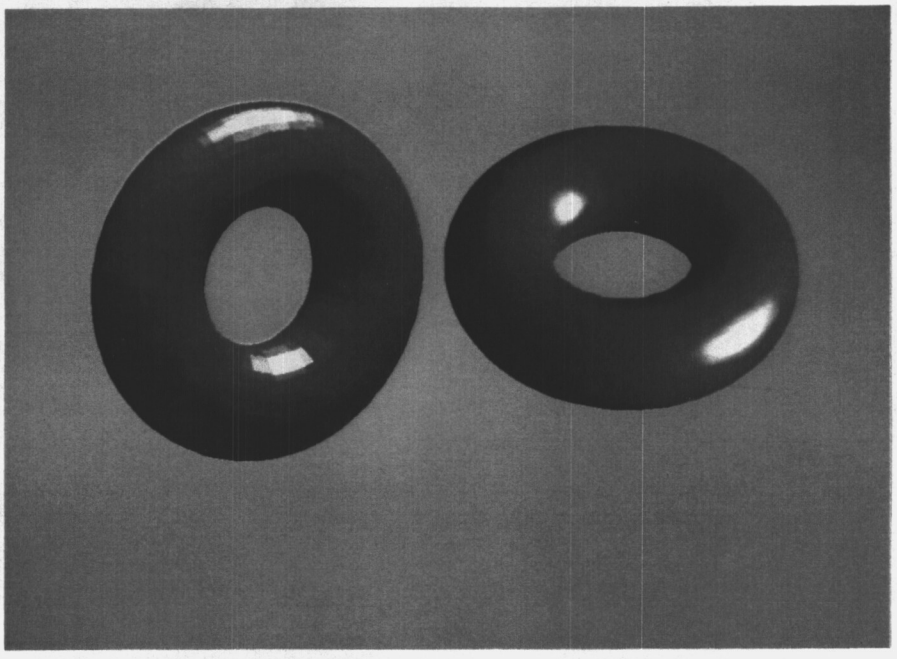


Figure 10: torus with and without Gouraud shading

workstation provides this algorithm in hardware.

The Gouraud shading can be kept local (within one patch) because the subdivision process results in smooth transitions between patches.

3.4 Hidden surface elimination & intersection of surfaces

The purpose of the hidden surface elimination is the removal of (parts of) surfaces that are not visible from the point of view. In case of intersecting surfaces (with possibly more intersection lines) only the visible parts must be displayed.

Originally a number of authors used scan line algorithms in order to perform the elimination of hidden surfaces for parametrically defined surfaces (see [Lane-80b]). A scan line algorithm consists of two nested loops. One for the Y-coordinate going down the screen and one for the X-coordinate going across each scan line of the current Y. Basically the scan line algorithm is as follows:

```
for each scan line y:
        for each pixel x on a scan line:
                for each surface intersecting that scan line at x:
                        calculate z-value
                determine the visible surface at x, y (lowest z-value)
```

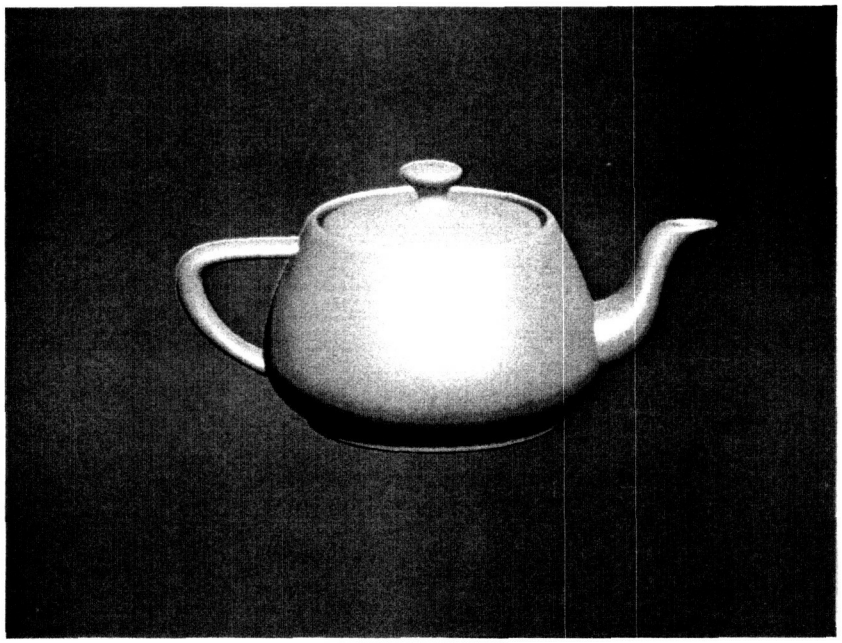


Figure 11: shaded model of "the teapot"

and display it.

For this implementation there has been chosen for a z-buffer hidden surface elimination algorithm. The scan line algorithms are rather time-consuming and besides that, the IRIS workstation provides a z-buffer algorithm in hardware.

The z-buffer algorithm has a number of advantages in relation to other hidden surface elimination algorithms. For example:

- the algorithm is simple,
- it can handle hidden surface elimination and complex surface intersections (scenes can be of any complexity),
- the increase of computing time with an increasing complexity of the scene is linear, and
- no sorting is necessary for determination which point has the lowest z-value.

The z-buffer algorithm needs two buffers. The video buffer is used to store the intensity of each pixel. The z-buffer is a separate buffer used to store the z-value or depth of every visible pixel. The depth or z-value of a new pixel to be written

to the video buffer is compared to the depth of that pixel stored in the z-buffer. If comparison indicates that the new pixel is in front of the pixel stored in the video buffer, then the new pixel is written to the video buffer and the z-buffer is updated with the new value. Otherwise, the new pixel and the corresponding z-value are discarded. The result of this algorithm is that only the visible pixels (surfaces) are displayed.

An example is shown in figure 11. It contains hidden surfaces and surfaces that are intersecting with each other (handle and sprout with the teapot itself, see figure 5). An example of a complex scene with z-buffering is shown in figure 6.

4 The Flexible Robot Arm POWER

4.1 Introduction

The Personal Occupied Woven Envelope Robot (POWER) project is a joint effort of the University of Alabama in Huntsville (UAH) and Wyle Laboratories [Wessling-86a] [Wessling-86b] [Wessling-87a]. This work is being performed under the Innovative Research Program for NASA.

POWER is a flexible robot arm. It will be used as an "extension" of the Space Station. POWER consists of 50 segments, and each segment has six degrees of freedom. The segments are based on the Stewart Table [Stewart-65], which has six linear (individually controlled) actuators. A control pod is attached to the top of the flexible robot arm. A flexible tunnel connects the control pod to the habitat module of the Space Station, allowing a person to transfer from the Space Station to the pod without having to suit up for extra vehicular activity. The operator of the pod is able to move himself and the pod to almost any location within 50 meters of the base attachment to the Space Station. The operator has at his disposal remote manipulator arms and also a glove box type arrangement with space suit arms so that he can perform manipulations on equipment external to the pod.

Some of the applications of POWER are [Wessling-86b]:

- Changing out and servicing payloads on the payload platform.
- Maintaining subsystems such as propulsion and attitude control.
- Providing satellite service.
- Performing inspections.
- Supporting shuttle cargo bay operations.
- Performing remote control operations for hazardous duty.
- Capturing satellites during final approach.

The robot arm is still under development. For graphical simulation and publicity purposes there was need for a graphical model of the flexible robot arm.

4.2 The graphical model of the flexible robot arm

The graphical model of the flexible robot arm consists basically of three elements, namely:

(1) *the wire-model*: Major parts of the Space Station are built up with the line-primitive (polygons).

(2) *the polygonal model*: The triangular segments of the robot arm are shaded polygons.

(3) *the piecewise Bezier surface*: The flexible robot arm itself and some parts of the Space Station (habitat module) are modeled by piecewise cubic Bezier surfaces.

In fact, these three elements belong to the boundary representation scheme discussed in chapter 1. Each part of the boundary representation scheme has its own advantages and typical applications. For example: (1) the Space Station itself is a wire model, (2) the triangular elements of the robot arm are flat-faced polyhedra, and (3) the remaining objects are free-form solids.

Only the Bezier representation of surfaces are covered in this report. The other techniques and some basic graphical concepts such as modeling and viewing transformations are assumed to be known by the reader and can be found in textbooks like [Newman-81], [Rogers-76] and [Rogers-85].

By modeling the Space Station as a wire frame, it is possible to do real time animation with the Space Station. When the animation stops, the shaded parts (habitat module and robot arm) are drawn with the hidden surfaces removed. This separation is necessary because of the limited capabilities of the IRIS workstation. There are only 32 bitplanes available. For real time animation two 16 bit-buffers are used (one for displaying and one for drawing, and vice versa). When the animation stops, one 16 bit-buffer becomes free and can be used for z-buffering. And besides the limited buffer space, the drawing of the shaded parts will take too long for real time animation.

Figure 12 depicts two segments of the flexible robot arm. One segment consists of two triangular plates connected by six individually controlled linear actuators (Stewart table).

Figure 13 and 14 show a close-up of a flexible robot arm with a few segments. One can see a part of the yellow habitat module (part of Space Station), the flexible robot arm and (on top of the robot arm) the control pod. A person can sit inside the control pod and look through the spherical window on top of the pod. Figure 14 shows the flexible transfer tunnel (red) for going from the Space Station to the control pod. For normal operation the flexible tunnel is retracted and fits in the yellow protection canister, placed at the first two segments of the robot arm.

Figure 15 and 16 display the total view of the flexible robot arm connected to the Space Station. The habitat module is located at the center of the Space Station.

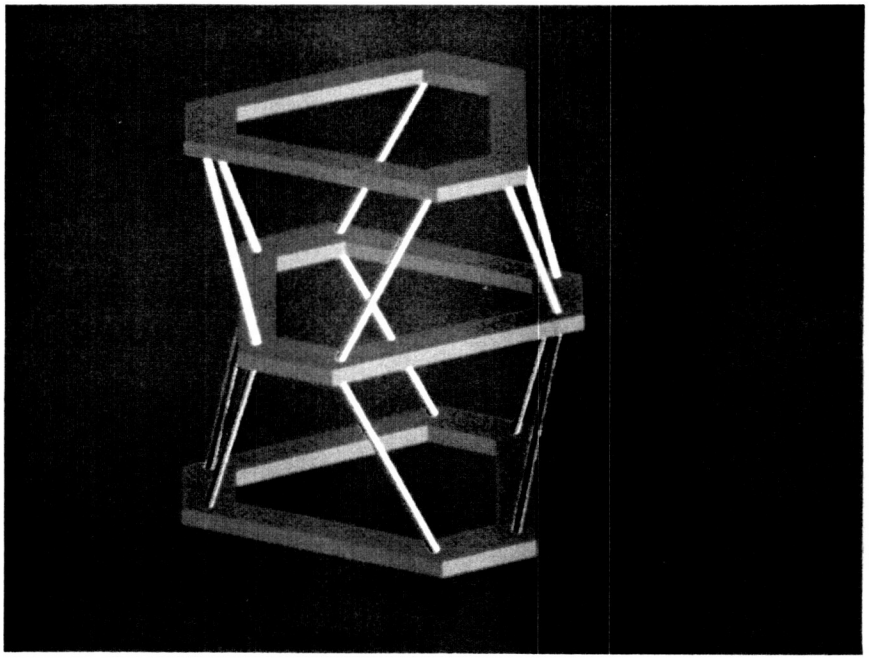


Figure 12: two segments of the flexible robot arm

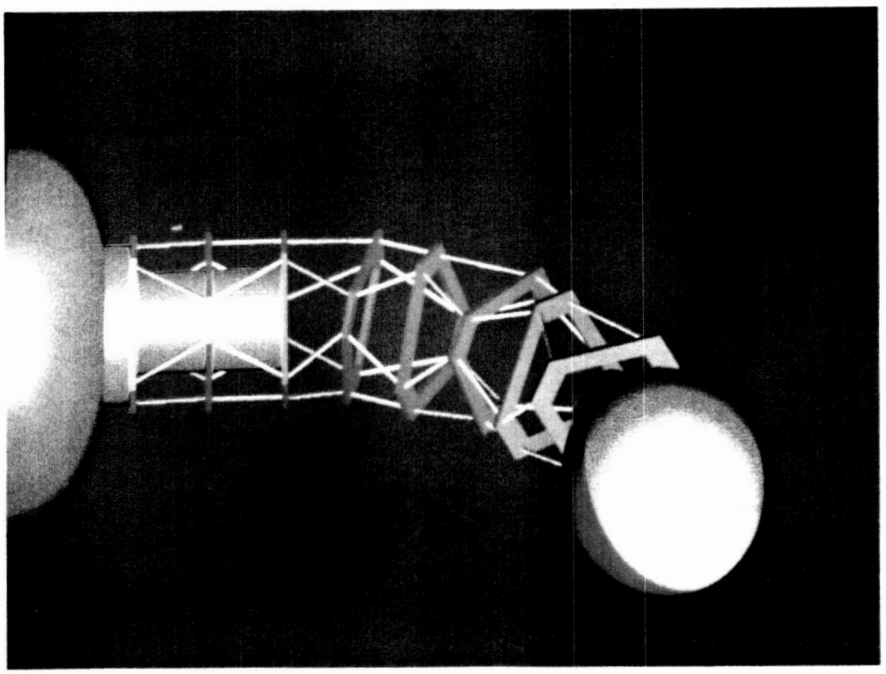


Figure 13: close-up of the flexible robot arm

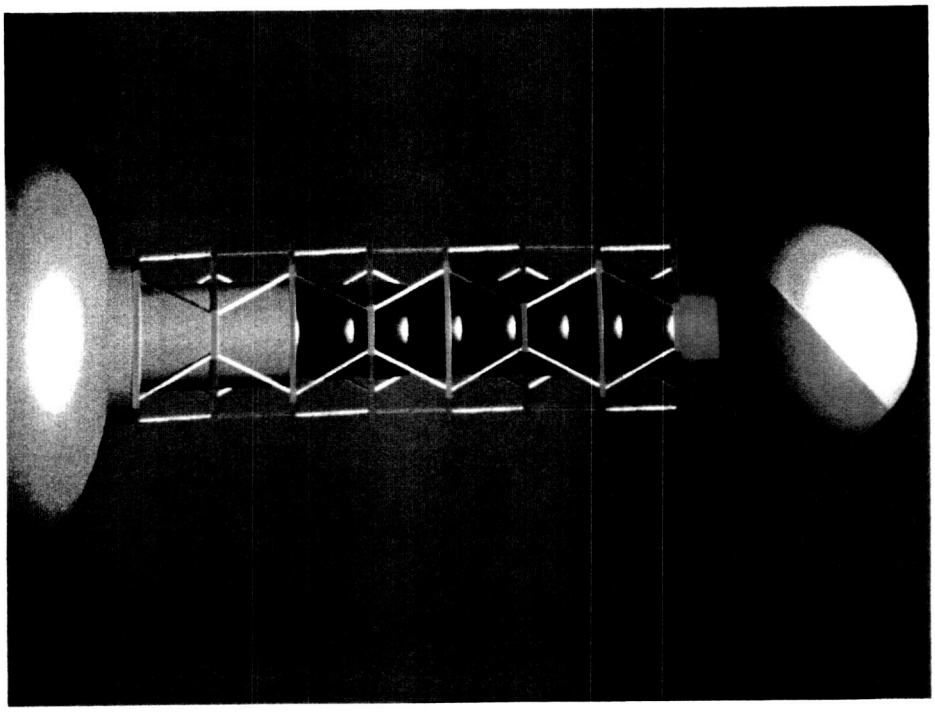


Figure 14: close-up of the flexible robot arm with the transfer tunnel

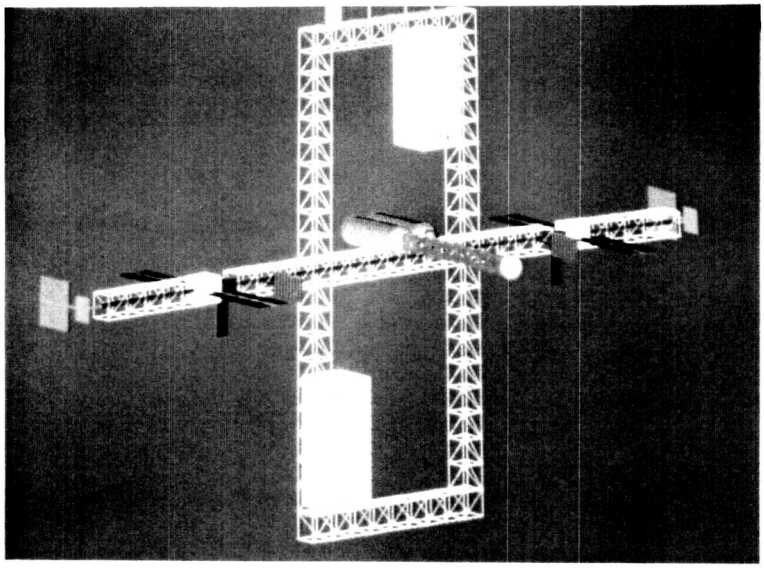


Figure 15: Space Station plus flexible robot arm (front view)

ORIGINAL PAGE
COLOR PHOTOGRAPH

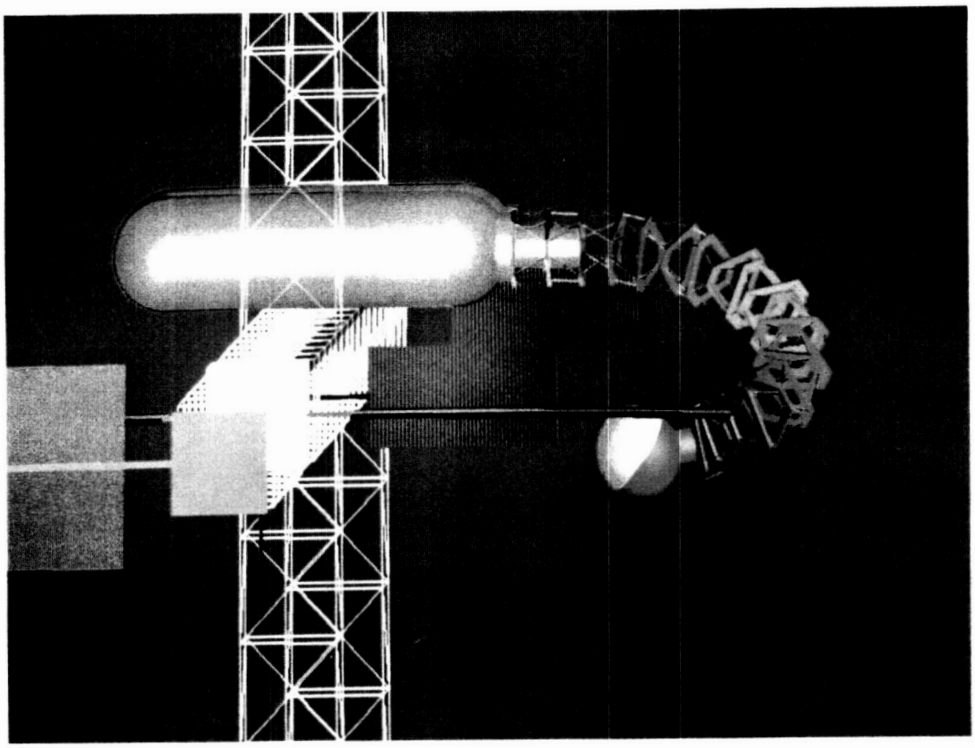


Figure 16: Space Station plus flexible robot arm (side view)

Conclusion

The purpose of this project was solid modeling for simulating the flexible robot arm POWER. In order to accomplish this goal, a general solid modeling software package for the IRIS workstation has been developed. The results are shown in figure 12, 13, 14, 15, and 16.

The Bezier formulation has been chosen for the representation of surfaces. This appears to be sufficient for modeling the robot arm and the Space Station.

If small patches are used, the surface is closely approximated by the recursive subdivision algorithm, and hence difficult time-consuming planarity estimations and integrity filters are not necessary. An important result of this research is the reformulation of the recursive subdivision algorithm of [Lane-80a]. The new algorithm is very elegant and gives a tremendous reduction of computing time, especially for hardware implementations.

The illumination model performs adequately for this purpose. The Gouraud shading gave the expected enhancement of the visual appearance of the surface on the screen.

Appendices

A The developed software package

The developed solid modeling package forms a layer on top of the software of the IRIS workstation. The IRIS workstation has no standard solid modeling primitives. The new software package enables the user (programmer) to display shaded polygon models and Bezier surfaces (with shading) on the screen. The software package is very general. Data, supplied by other authors ([Crow-87] and [Blinn-87]), can be displayed on the screen without any problem.

In addition to the solid modeling package a number of application programs are written. First of all, programs are written for generating the Space Station and the flexible robot arm. Furthermore there are some demonstration programs written for illustrating the possibilities of the solid modeling package.

In the remainder of this section a list with the most important programs and procedures are given. The programming language is *C*.

Solid modeling package

CalcNorm This procedure calculates the normals of a shaded polygon model. In this case one segment of the flexible robot arm.

Crossp This procedure determines the cross product of two vectors. The resulting vector is normalized.

DefMatrix This procedure defines a $4 * 4$ matrix, which performs a certain translation and rotation.

DrawPatch This procedure draws the convex hull of a Bezier surface patch on the screen. Each face of the convex hull has one shade.

DrawSeg This procedure draws a shaded polygon model on the screen. In this case one segment of the flexible robot arm.

FindIntens This procedure calculates the light intensity out of the normal, light, and eye vector.

Gensurf This procedure generates the surface of a patch by calling procedure Split and recursively calling procedure Gensurf for each subpatch.

Gouraud This procedure draws a convex hull of a Bezier surface patch on the screen, with Gouraud shading applied.

Load This program reads a file from disk and displays it on the screen.

LoadMap This procedure loads the lookup table with shades of some predefined colors.

PolyTrans This procedure multiplies coordinates with the matrix defined by procedure DefMatrix.

Save This program stores the picture on the screen into a file on disk.

Size This procedure determines the size of a patch.

Split This procedure does the splitting of the surface into 4 subpatches.

Application programs

Robot This program generates a close-up of the flexible robot arm.

Station This program generates the Space Station and the flexible robot arm, and displays them on the screen.

Demonstration programs

Glasses This program displays the six wine glasses on the screen.

Patchnet This program displays a Bezier surface patch with curves of constant parameter and the corresponding mesh of control points.

Sweep This program generates a picture of a solid swept along a curve.

Teapot This program displays the wire and solid model of the teapot on the screen.

Testsurf This program displays the surface of Program Patchnet with shading.

Torus This program displays the torus with and without Gouraud shading.

Tube This program displays two tubes on the screen. One with and the other without highlighting.

B The correct subdivision algorithm

In [Lane-80a] an algorithm for subdivision of surfaces is given. The described algorithm has typos and missing statements. In this appendix the correct algorithm is described.

Given the polynomial coefficients $P \equiv P_{ij}$, $i = 0, 1, \ldots, m$; $j = 0, 1, \ldots, n$ in terms of a Bernstein basis on $[0,1] \times [0,1]$. Compute the subpatches $Q \equiv Q_{ij}$, $R \equiv R_{ij}$, $S \equiv S_{ij}$, and $T \equiv T_{ij}$.

Procedure Surf_Split (P, Q, R, S, T, m, n)

step 1: [initialize] Set $Q \leftarrow P$

step 2: [split in u direction]
for $k = 0$ **to** n
begin
$R_{m,k} \leftarrow Q_{m,k}$
for $p = 1$ **to** m
begin
$QTMP2 \leftarrow Q_{p-1,k}$
for $q = p$ **to** m
begin
$QTMP1 \leftarrow QTMP2$
$QTMP2 \leftarrow (Q_{q-1,k} + Q_{q,k})/2$
$Q_{q-1,k} \leftarrow QTMP1$
end
$Q_{m,k} \leftarrow QTMP2$
$R_{m-p,k} \leftarrow QTMP2$
end
end

step 3: [split Q in v direction]
for $k = 0$ **to** m
begin
$S_{k,n} \leftarrow Q_{k,n}$
for $p = 1$ **to** n
begin
$QTMP2 \leftarrow Q_{k,p-1}$
for $q = p$ **to** n
begin
$QTMP1 \leftarrow QTMP2$
$QTMP2 \leftarrow (Q_{k,q-1} + Q_{k,q})/2$
$Q_{k,q-1} \leftarrow QTMP1$

```
                end
                Q_{k,n} ← QTMP2
                S_{k,n-p} ← QTMP2
            end
        end
step 4: [split R in v direction]
        for k = 0 to m
        begin
            T_{k,n} ← R_{k,n}
            for p = 1 to n
            begin
                RTMP2 ← R_{k,p-1}
                for q = p to n
                begin
                    RTMP1 ← RTMP2
                    RTMP2 ← (R_{k,q-1} + Q_{k,q})/2
                    R_{k,q-1} ← RTMP1
                end
                R_{k,n} ← RTMP2
                T_{k,n-p} ← RTMP2
            end
        end
        return
```

C Calculation of subdivision matrices

The calculation of the matrices S', S'', S''', and S''''. See paragraph 2.3.3.

$$J = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$J^{-1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1/3 & 1 \\ 0 & 1/3 & 2/3 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$S = \begin{pmatrix} 1/8 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S' = J^{-1}SJ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 1/4 & 1/2 & 1/4 & 0 \\ 1/8 & 3/8 & 3/8 & 1/8 \end{pmatrix}$$

$$S'' = S'^{*}I^{*} = \begin{pmatrix} 1/8 & 3/8 & 3/8 & 1/8 \\ 0 & 1/4 & 1/2 & 1/4 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S''' = S'^{t} \begin{pmatrix} 1 & 1/2 & 1/4 & 1/8 \\ 0 & 1/2 & 1/2 & 3/8 \\ 0 & 0 & 1/4 & 3/8 \\ 0 & 0 & 0 & 1/8 \end{pmatrix}$$

$$S'''' = S''' * I^* = \begin{pmatrix} 1/8 & 0 & 0 & 0 \\ 3/8 & 1/4 & 0 & 0 \\ 3/8 & 1/2 & 1/2 & 0 \\ 1/8 & 1/4 & 1/2 & 1 \end{pmatrix}$$

References

[Amanatides-87] *Realism in Computer Graphics: A survey*
J. Amanatides
IEEE on Computer Graphics and Applications, Jan. 1987

[Blinn-87] *Displays on display: The Doughnut-data*
J. Blinn
IEEE on Computer Graphics and Applications, April 1987

[Crow-87] *The Origins of the Teapot*
F. Crow
IEEE on Computer Graphics and Applications, Jan. 87

[Eastman-84] *A Review of Solid Shape Modelling based on integrity verification*
C. M. Eastman and K. Preiss
Computer Aided Design, vol. 16, no. 2, March 1984

[Gouraud-79] *Computer Display of Curved Surfaces*
Thesis, University of Utah, 1971
H. Gouraud
Garland Publishing Inc, New York, 1979
ISBN 0-8240-4412-6

[Lane-80a] *A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces*
J. M. Lane and R. F. Riesenfeld
IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-2, no. 1, Jan. 1980

[Lane-80b] *Scan Line Methods for Displaying Parametrically Defined Surfaces*
J. M. Lane, L. C. Carpenter and J. F. Blinn
Communications of the ACM, vol. 23, no. 1, Jan. 1980

[Lorig-86] *Advanced Image Synthesis-Shading*
G. Lorig
Advances in Computer Graphics I
edited by G. Enderle, M. Grave, and F. Lillehagen

Springer Verlag, Germany, 1986
ISBN 3-540-13804-8

[Mortenson-85] *Geometric Modeling*
M. F. Mortenson
John Wiley & Sons, U.S.A., 1985
ISBN 0-471-88279-8

[Mudur-86] *Mathematical Elements for Computer Graphics*
S. P. Mudur
Advances in Computer Graphics I
edited by G. Enderle, M. Grave, and F. Lillehagen
Springer Verlag, Germany, 1986
ISBN 3-540-13804-8

[Newman-81] *Principles of Interactive Computer Graphics*
W. M. Newman and R. F. Sproull
McGraw-Hill Inc., U.S.A., 1981
ISBN 0-07-046338-7

[Pulleyblank-87] *The feasibility of a VLSI Chip for Ray Tracing Bicubic Patches*
R. Pulleybank and J. Kapenga
IEEE on Computer Graphics and Applications, March 1987

[Robertson-87] *Pushing the Limits of 3D Technology*
B. Robertson
Computer Graphics World, April 1987

[Requicha-80] *Representations for Rigid Solids: Theory, Methods, and Systems*
A. A. G. Requicha
Computing Surveys, vol. 12, no. 4, Dec. 1980

[Requicha-81] *An Introduction to Geometric Modeling and its Applications in Mechanical Design and Production*
A. A. G. Requicha and H. B. Voelcker
Advances in Information Systems Science, vol. 8, 1981, edited by ulius T. Tou (Plenum Publishing Company)

[Requicha-83] *Solid Modeling: Current Status and Research Directions*
A. A. G. Requicha and H.B. Voelcker
IEEE Computer Graphics and Applications, Oct. 1983

[Rogers-76] *Mathematical Elements for Computer Graphics*
D. F. Rogers and J. A. Adams
McGraw-Hill Book Company, U.S.A., 1976
ISBN 0-07-053527

[Rogers-85] *Procedural Elements for Computer Graphics*
D. F. Rogers
McGraw-Hill Book Company, U.S.A., 1985
ISBN 0-07-Y66503-6

[Stewart-65] *A Platform with Six Degrees of Freedom*
D. Stewart
Proc. Inst. Mechanical Engineering (London), vol. 180,
no. 15, 1965

[Tamminen-85] *An Integrity Filter for Recursive Subdivision Meshes*
M. Tamminen, F. W. Janssen
Computers & Graphics, vol. 9, no. 4, pp. 351-363, 1985

[Wessling-86a] *Progress Report: Personnel Occupied Woven Envelope Robot*
F. C. Wessling, G. B. Bakken, W. Teoh, M.C. Ziemke, V. Patel
Johnson Research Center (UAH) & Wyle Laboratories,
Nov. 30, 1986

[Wessling-86b] *Personnel Occupied Woven Envelope Robot*
F.C. Wessling, W. Teoh, and M.C. Ziemke
Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, U.S.A., Nov. 1986

[Wessling-87a] *Progress Report: Personnel Occupied Woven Envelope Robot*
F. C. Wessling
Johnson Research Center (UAH), June 1, 1987

[Wessling-87b] *Final Report: Personnel Occupied Woven Envelope Robot*
F. C. Wessling et al.
Johnson Research Center (UAH), to be published in Fall 1987

APPENDIX B

GOAL DRIVEN KINEMATIC SIMULATION OF FLEXIBLE ARM ROBOT FOR SPACE STATION MISSIONS*

Paul J. Janssen

July 26, 1988

Center for Applied Optics
University of Alabama in Huntsville
Huntsville, AL 35899 USA

* Work supported by NASA Grant #NAGW-847
and State of Alabama Research Council

PRECEDING PAGE BLANK NOT FILMED

PREFACE

Flexible arms offer a great degree of flexibility for maneuvering in space environment. The problem of a Space Station based Flexible Arm Robot has been studied. A computer simulation software package including control algorithms has been developed and implemented in the C programming language on a Silicon Graphics Workstation 3020 with a UNIX operating system.

The Personnel Occupied Woven Envelope Robot (POWER) is a joint project of the Johnson Research Center of the University of Alabama in Huntsville (UAH), Pace & Waite Incorporated, and Wyle Laboratories.

The author wishes to acknowledge the following people for their contributions to this work.

Dr. Francis C. Wessling, Principal Investigator of the POWER project, Associate Director of the "Consortium for Materials Development in Space" at UAH, for his guidance and support.

Dr. Amar Choudry, Senior Research Scientist at the Center for Applied Optics (CAO), UAH, (Adjunct) Professor Computer Science at UAH, for his support.

Dr. B. Schroer, Director of the Johnson Research Center, UAH, for use of facilities and equipment.

Vincent J. Harrand, Graduate Research Assistant, CAO-UAH, for designing a graphical wire model of the Space Station.

My wife **Katherin**, for proof-reading this report.

Dr. Ir. J. Smith, visiting professor at CAO-UAH, Senior Research Scientist at the Academical Medical Center / University of Amsterdam (AMC/UvA), for his final commentary.

Paul J. Janssen, July 26, 1988.

TABLE OF CONTENTS page#

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

1. INTRODUCTION

Working in space environment can be very stressful and time consuming. Approximately 75 percent of an astronaut's time spent on extra vehicular activities (EVA) is wasted by suiting up and undressing. Because the astronaut is wearing a space suit, simple muscle contractions, for example of the hand, take considerably more force and energy. The concept of the Personnel Occupied Woven Envelope Robot (POWER) was developed for reaching any location around the Space Station without performing EVA (figure p1). POWER is also known by the name of Flexible Arm Robot (FAR) and Human Occupied Space Teleoperator (HOST).

The Flexible Arm Robot is a tube-shaped robot, built out of several stacked, individually controllable segments (figure p2). The bottom segment is attached to the Habitat Module of the Space Station. The top segment is connected to a control pod, in which a human operator will command FAR, enabling the operator to move himself and the control pod to a desired location around the Station. The pod's control mechanisms and large transparent window make it unnecessary for the operator to rely on television screens to see his motions or to perform his work. Each segment has six degrees of freedom and is based on the 'Stewart Table'. This model was introduced by D. Stewart in 1965 [Stewart65] and has been used successfully for flight simulator platforms.

Connected to the pod are two three-link remote manipulator arms, and a glove box type arrangement with space suit arms, for the more detailed work outside the pod. An external toolbox allows the operator to change manipulators on the remote arms without having to return to the Habitat Module.

When FAR is in retracted position, a transfer tunnel through the segments connects the Habitat Module to the control pod. The astronaut can climb through this tunnel from the Habitat Module to the pod, eliminating the necessity of wearing a space suit. The access tunnel has undergone some modification since the original concept was presented. It was originally thought that the tunnel might serve as a structural component. However, the possible penetration by micro-meteoroids and subsequent loss of pressure changing its rigidity caused a reevaluation of its function. The tunnel now serves solely as an access way to the pod, and will be retracted when FAR is in operation.

Structural analyses demonstrated that the chosen design of the Flexible Arm Robot has only sufficient strength for the operation of a four segment prototype in a one gravity environment [Wessling86]. New control-algorithms to handle this system of large degrees of freedom are needed. Therefore, a computer simulation of the whole Flexible Arm Robot, containing up to fifty segments, is desired. For this purpose, a kinematic model based on the physical features should be developed. Chapter 2 describes the physical model. Chapter 3 presents the graphical design for the simulation studies. In chapter 4 the kinematic equations are specified, for implementation in the motion algorithms presented in chapter 5.

2. PHYSICAL DESIGN

FAR consists of n stacked segments, each based on the Stewart table. A segment consists of two plates, six linear actuators and twelve joints (figure p4). A plate is a triangular shaped hexahedron with a length of 92.464 cm (36.403 in.), a width of 6.2548 cm (2.4625 in.) and a height of 7.620 cm (3.000 in.). An actuator is a extendable and retractable link between the two plates. In retracted position, the actuator is 50.541 cm (19.898 in.) long. A fully extended actuator is 81.021 cm (31.898 in.) long (figure p5). A joint is a two-axis connection between a plate and an actuator and is located at each vertex of each plate. The upper (maneuverable) plate and the lower (base) plate are rotated 180 degrees in respect to each other. The maneuverable plate of a particular segment forms the base of the next segment, while its own base is the maneuverable plate of the previous segment (figure p3).

2.1. Actuators

The actuators could be hydraulic, electric, or pneumatic. Hydraulic actuators require bulky high pressure hoses and well lubricated pistons. Seals tend to be a problem. Leakage of hydraulic oil was deemed a real possibility unless precautions were taken to protect the hoses from micro-meteoroid impact. This would add to the bulk of the system, consequently, a hydraulic based system was discarded. Pneumatic systems require supply and return hoses too. in addition, pneumatic systems can leak, and require more power than electrical or hydraulic systems. Thus, an electrically based system was chosen. Wires can be smaller than hoses, cause no loss of hydraulic fluid or air, and can be fused for safety. Threading wire around the structure or exoskeleton should cause no large difficulty. Electric motor driven actuators are used successfully on the Remote Manipulator System. These were specially designed for space use. That technology may be directly usable in this application.

2.2. Joints

A typical Stewart table is usually shown with ball-joints at each side of the actuator. However, ball-joints have some disadvantages in this application. Ball-joints on both ends of the actuator do not allow very much torque to be applied by electrical motors driving the screw of the actuator. Torque can only be applied if it is less than the torque that causes the ball joint to slip in its mounting. Ball joints-also allow uncontrolled rotation of the two ends of the actuator. Thus, ball joints were rejected. Instead, a double clevis connector was designed. This connector allows angular motion of each of the actuators but restricts the rotation of the ends of the actuator with respect to the vertices of the Stewart table. Consequently, free motion is attainable without the danger of the wires wrapping around the actuators.

2.3. Transfer Tunnel

The transfer tunnel started as a continuous tunnel spanning from the Habitat Module to the control pod. Moving the tunnel requires work unless the motion is accomplished under constant volume conditions or unless the tunnel is evacuated. An airlock is planned at both ends of the tunnel to isolate both the Habitat Module and the control pod from the tunnel. Thus, evacuation of the air from the tunnel is possible and would remove the compression power requirements. Since the air locks are on both ends of the tunnel, it can be retracted when FAR is in operation, which reduces the likelihood of puncture by micro-meteoroids. A mechanism is required to extend and retract the tunnel. One successful application of a flexible tunnel in space is the access tunnel used between the Orbiter and Spacelab. Other applications, including a one person expandable airlock, have been suggested by Goodyear Aerospace Corporation.

One might question whether an access tunnel is necessary, or whether access to the pod can be possible without it. Let us assume that the boom or exoskeleton of FAR has an extended range of fifty meters and an expansion ratio of five to one. Thus, its retracted length would be ten meters. This requires an access tunnel of ten meters. A fixed tunnel projecting ten meters from the Habitat Module, would require ten meters of exoskeleton dead length to accommodate it. An expandable, retractable tunnel with a five to one expansion ratio would require only two meters of dead length in the exoskeleton and need only two meter storage length for the access tunnel, and would be completely out of the way for motion of the Flexible Arm Robot.

The above considerations assume FAR's bottom segment is attached to the Habitat Module of the Space Station. If the base were to be connected to some other part of the Space Station, it might be possible to couple the airlocks of the Habitat Module and the control pod directly. The astronaut could then climb from the Habitat Module to the control pod, eliminating the necessity of a transfer tunnel.

2.4. Control Pod

An environmental control and life support system (E.C.L.S.S.) for the control pod could be self contained or attached to the Habitat Module. Being self contained does not mean it has to be self perpetuating. Consumables could be replenished from the module. Thus, a system similar to that used in the space suit appears to be a likely candidate for an ECLSS for the control pod. It appears to be small enough for an astronaut to carry it through the access tunnel to and from the pod, yet large enough to allow approximately eight hours of operation in the pod for each backpack that the astronaut carried to the pod. Johnson Space Center is developing a regenerative backpack that would free the astronaut from transporting one. Other aspects of a total ECLSS would include passive measures in the pod such as good insulation, proper materials, sun shielding in the viewing port, and so forth. Attaching an ECLSS directly to the Habitat Module does not appear to be practical for several reasons.

An ECLSS dependent directly on the ECLSS of the Habitat Module would require a reevaluation and possible redesign of the ECLSS for the module because of the additional loads imposed by the pod. This does not appear to be desirable. In addition, connecting supply lines between the control pod and the module exposes the module to loss of pressure caused by the rupture of the supply lines to the pod. Check valves could be installed to prevent large losses. Other considerations, as well, dictate that an independent life support system be used. Running the supply lines along a moving structure increases the possibility of accidental cut. The lines also require an extra micro-meteoroid protection. Having the ECLSS independent of the module appears to be the best way to avoid many of the potential problems, particularly when one recognizes that an independent ECLSS has already been developed for the space suit [Wessling86].

3. GRAPHICAL DESIGN

The main objective of the Flexible Arm Robot simulation is verification of the kinematic model, presented in chapter 4. It is desirable to see an interactively controlled simulation on a graphical display representing the dynamic scenes, rather than examining an endless stream of numbers. Thus, a graphical model is required. A wire frame will suffice and will take relatively little time to draw, which is imperative for real-time motion.

3.1. Plates

A plate is defined by the coordinates of its six vertices. Six lines connect these vertices to form a regular hexahedron, representing the plate. Origin O of the x,y,z-axes is in the center of the plate. Constant value r is the distance between origin O and each vertex. Constant value beta is the angle between the line from O to a vertex of the plate and the line from O to the adjacent vertex of the imaginary triangle (figure g1). Since only length, width and height of a plate are known, beta and r must be computed.

$$
\begin{aligned}
3b \,/\, (L + 2W) &= \text{sqrt}[\,3\,] \,/\, 2 \\
b &= (L + 2W) * \text{sqrt}[\,3\,] \,/\, 6 \\
r^2 &= b^2 + (L/2)^2 \\
&= (L + 2W)^2/12 + 3L^2/12 \\
&= (\,L^2 + 4W^2 + 4LW + 3L^2\,) \,/\, 12 \\
&= (\,L^2 + W^2 + LW\,) \,/\, 3 \\
\text{beta} &= \arcsin(\,W \,/\, 2r\,)
\end{aligned}
$$

Algorithm 3.1 is used to compute the coordinates of the six vertices. The x-axis is assumed through the middle of vertices 1 and 2. The y-axis points upward, parallel to the line connecting vertices 4 and 5.

```
(1)     FOR i=0 TO 5 DO
(2)         sign    =   ( (i & 1) << 1 ) - 1
(3)         angle   =   ( i DIV 2 ) * PI * 2/3  +  beta * sign
(4)         B[i+1].x =  r * COS( angle )
(5)         B[i+1].y =  r * SIN( angle )
(6)         B[i+1].z =  0
(7)     ENDDO
```

The loop body between lines (1) and (7) is executed six times. Each loop iteration the x,y,z-coordinates of vertex i+1 of segment u (B_i) is computed. Figure g1 shows the imaginary equilateral triangle through the six vertices. The first two loop iterations are based on the angle between x-axis and the line from origin 0 to the triangle vertex near the plate vertices 1 and 2. This angle is equal to zero. The first loop iteration subtracts beta degrees from this angle, causing the line to cross vertex 1. The second iteration adds beta degrees to this angle, causing the line to cross vertex 2. ~~Sixty~~ 120 degrees are added to the line for iterations three and four. It now runs from origin 0 to the triangle vertex near plate vertices 3 and 4. Again, beta degrees subtracted causes the line to cross vertex 3, beta added crosses vertex 4. The last two iterations behave similarly. Obviously, beta degrees are subtracted for odd and added for even vertices.

Line (2) determines the sign (add or subtract). Vertices 1 and 2 correspond to i=0 and i=1 respectively. The (i & 1) is a bitwise 'and' operation, yielding 0 for odd, 1 for even vertices. The (x << 1) is a left-shift of 1 bit, equivalent to a multiplication by 2, yielding 0 or 2. Finally, the intermediate result is decreased by one, yielding -1 or 1. Thus, the value of variable 'sign' will be -1 for odd and +1 for even vertices.

Line (3) determines the angle of the line from origin 0 through one of the vertices of the triangle. The angle (or line) must be equal for vertex 1 and 2, for vertex 3 and 4, and for vertex 5 and 6. The operation 'div' refers to integer division, so that '3 div 2' gives 1. The expression (i div 2) yields 0 for vertices 1 and 2 (i=0 and i=1), yields 1 for vertices 3 and 4, and 2 for vertices 5 and 6. Consequently, variable 'angle' will be 0, 1 or 2 times ~~60~~ 120 degrees (PI*~~1/3~~ 2/3 radial). Finally, the desired angle between x-axis and the line from origin 0 to vertex i+1 is obtained by adding sign times beta (+beta or -beta).

Lines (4), (5) and (6) compute the x,y,z-coordinates respectively. B_i represents the coordinates of vertex i (figure g3). Since the origin of the coordinate-axes is assumed in the center of the plate, with the z-axis pointing upward, the z-coordinates are zero.

3.2. Segments

An important observation is that both maneuverable and base plates are identical graphical objects. Therefore, the maneuverable plate can be derived from the base plate. This can be done by rotating it 180 degrees around the z-axis. Now the DOF variables can be used to give the plate its final position (figure g2). Because of the 180 degree z-rotation, vertex 1 of the maneuverable plate is derived from vertex 4 of the base plate, vertex 2 from vertex 5, etc. In general, vertex i of the maneuverable plate is derived from vertex [(i+2) mod 6] + 1 of the base plate. The maneuverable plate could also be derived by applying a 30 degree z-rotation. In fact, only the combination z-rotation with vertex-mapping is important.

3.3. Flexible Arm

The particular configuration of FAR depends on the configuration of each of the n segments. The configuration of a segment depends on the values of the six degrees of freedom, the DOF variables (figure g2). For each segment u ($1 \le u \le n$), the DOF variables define the position and orientation of the maneuverable plate, relative to the center of its base. The DOF variables are denoted by the six tuple D_u. Elements 1,2,3 corresponds to the x,y,z-translation, elements 4,5,6 to the x,y,z-rotation respectively. B_i are the coordinates of vertex i of the base plate (figure g3). The coordinates B are constant and defined in section 3.1. The coordinates of the maneuverable plate's vertex i is called $M_{u,i}$ (figure g4). The lengths of segment u's actuators are denoted by the six tuple A_u (figure g5). Element $A_{u,i}$ ($1 \le j \le 6$) corresponds to the length of actuator i, which is connected to vertex i of the base and vertex i of the maneuverable plate.

The following algorithm is used to draw the Flexible Arm Robot, based on the values of the DOF variables.

```
(1)  FOR u=1 TO n DO
(2)       DrawPlate( B1 , B2 , B3 , B4 , B5 , B6 )
(3)       Tu = TransMatrix( Du,1, Du,2, Du,3, Du,4, Du,5, Du,6+ 180 )
(4)       FOR i=1 TO 6 DO
(5)            Mu,i = Bi  . Tu
(6)       ENDDO
(7)       FOR i=1 TO 6 DO
(8)            Au,i = | Bi - Mu,1+(i+2)%6 |
(9)            DrawActuator( Au,i )
(10)      ENDDO
(11)      MoveXYZaxes( Tu )
(12) ENDDO
(13) DrawPlate( B1 , B2 , B3 , B4 , B5 , B6 )
```

Algorithm 3.3. The number of plates to be drawn in a n segment Flexible Robot Arm is equal to n+1. Since all plates are identical graphical objects, only one function (DrawPlate) is needed to draw a plate, based on the values of B_i ($1 \leq i \leq 6$). Important is the orientation and position of the XYZ-axes, relative to which the plates are drawn. The index u is used wherever the variable depends on the current segment.

The main loop is from line (1) to line (12). Segment u is the current segment to be drawn. For the first iteration (u=1), the coordinate axes are assumed to be at the correct position on the Habitat Module of the Space Station. For the next iterations (u=2..n) the coordinate axes will be in the center of segment u's base plate, which is the maneuverable plate of segment u-1 (see line 11).

Line (2) draws the base plate of segment u. Segment u's maneuverable plate will not be drawn. This is done during the next loop iteration when segment u+1's base plate is drawn.

Line (3) defines the transformation matrix T_u. T_u is a 4x4 matrix, dependent on the DOF variables D_u of segment u. Its definition will be described in detail in section 4.2. The z-rotation $D_{u,6}$ is incremented with 180 degrees, and the corresponding mapping (line 8) will be used.

Line (4), (5) and (6) compute the vertices of the maneuverable plate of segment u, using the coordinates of the base plate and transformation matrix T_u. Line (5) cannot be integrated in the loop from (7) to (10), for the vertex mapping described in section 3.1 requires that all vertices first be computed.

Line (8) computes the length of actuator $A_{u,i}$, which is the distance between vertex i of the base and vertex i of the maneuverable plate. The mapping is used in combination with the 180 degree z-rotation in line (3). Line (9) draws the actual actuator. The loop from (7) to (10) makes sure that six actuators are drawn per segment.

Line (11) repositions the coordinate axes (see figure g2, maneuverable plate) using the transformation matrix T_u, which is based on D_u.

The algorithm can be seen as successive drawings of base plates. Since segment n+1 does not exist, its base plate - and segment n's maneuverable plate - will not be drawn. Line (13) draws this additional plate.

The actuators are the only parts of the configuration with a changeable length. Consequently, their lengths must be checked during motion to prevent them from exceeding their minimum or maximum values. An other constraint involves the angle between actuator and plate. If this angle gets too small, the forces on the joint could cause it to collapse. Manual control during simulation could maneuver a segment in such a position that one or more actuators breakdown. This situation is visualized by changing the color of the(se) actuator(s) on the computer screen, while halting motion and restoring the last legal configuration.

4. KINEMATICS

4.1. Control Strategies

Since the Flexible Arm Robot consists of n segments, 6n degrees of freedom must be controlled. Three control strategies have been considered to date: the tip-biased, the base-biased and the equal-biased methods.

In the tip-biased strategy, more preference is given to move those segments that are closest to the control pod. The main advantage of this scheme is that less mass must be moved at any given time, thereby minimizing the power consumption. The main disadvantage of this scheme involves the dynamics. Consider an almost fully extended configuration, describing a curve. Changing the length of an actuator of the base segment by a tenth of an inch, could cause the control pod to move several feet.

The base-biased strategy is similar to the tip-biased one, except that preference is given to those segments that are closest to the base of the Flexible Arm Robot. The main disadvantage of this scheme is that any movement involves the entire column.

The equal-biased strategy emphasizes equality for each segment, so that the configuration of all segments is the same. It introduces a difference between even and odd segments. All even segments are rotated 180 degrees in respect to the odd segments, so the sign of their x and y rotations and translations differ. An advantage of the equal-biased scheme is the equal wear of all parts, but the main disadvantage is the reduced flexibility, from 6n to 6 degrees of freedom.

4.2. Direct Kinematics

Data driven motion is achieved by explicitly changing the variables which define the configuration of the Flexible Arm Robot. Since the actuators are its only variable parts, the first way to define a particular configuration is by all the actuator lengths $A_{u,i}$ ($1 \le u \le n$, $1 \le i \le 6$). A second way the Arm's configuration is defined is by all the DOF variables $D_{u,i}$ ($1 \le u \le n$, $1 \le i \le 6$). Each set of DOF variables is related to one and only one set of actuator lengths.

Manipulation of both the actuator lengths and the DOF variables are desired. When the DOF variables are modified, the new actuator lengths must be calculated. This process will be called 'Actuator extension transformation' (section 4.2.2). When the actuator lengths are changed, the new DOF variables must be calculated. This process will be called 'Inverse actuator extension transformation' (section 4.2.3). The following sections will illustrate how to derive these sets of variables from each other.

4.2.1. List of Variables

The relation actuator length and DOF variables involve only actuators and DOF variables of the same segment. Therefore, the equations will focus on one individual segment, so that the index u can be omitted.

A_i Length actuator i ($1 \le i \le 6$).

D DOF variables vector.

$$(D_1 , D_2 , D_3 , D_4 , D_5 , D_6)^T$$

M_i Coordinates vertex i of maneuverable plate ($1 \le i \le 6$).

$$(M_{i1} , M_{i2} , M_{i3} , 1)^T$$

B_i Coordinates vertex i of base plate ($1 \le i \le 6$).

$$(B_{i1} , B_{i2} , B_{i3} , 1)^T.$$

[T] Transformation matrix T.

$$\begin{pmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ T_{41} & T_{42} & T_{43} & T_{44} \end{pmatrix}$$

C_i Vector from B_i to M_i ($1 \le i \le 6$).

$$(C_{i1} , C_{i2} , C_{i3} , 0)^T$$

[J] Jacobian with partial derivatives $J_{pq} = \partial F_p / \partial D_q$ ($1 \le p,q \le 6$).

$$\begin{pmatrix} J_{11} & J_{12} & \cdots & J_{16} \\ J_{21} & J_{22} & \cdots & J_{26} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ J_{61} & J_{62} & \cdots & J_{66} \end{pmatrix}$$

F Function vector for Newton-Raphson approximation.

$$(F_1 , F_2 , F_3 , F_4 , F_5 , F_6)^T$$

4.2.2. Actuator Extension Transformation.

A segment is driven by actuator extensions. When motion is desired in each degree of freedom, a transformation to actuator extensions is necessary. The objective of this section is to compute the actuator lengths A_i ($1 \le i \le 6$) given the DOF variables D_i ($1 \le i \le 6$).

The desired transformation is closely related to the coordinate transformations needed for the graphical representation of the Flexible Arm Robot as described in section 3.3.

(1) compute [T]

(2) FOR i=1 TO 6 DO

(3) $M_i = [T]^T \cdot B_{1+(i+2)\%6}$

(4) $A_i = | M_i - B_i |$

(5) ENDDO

Line (1) of algorithm 4.2.2 computes the (4x4) transformation matrix [T] (see 4.2.1.) using the DOF variables D. The constant coordinates B of the base plate are assumed to be defined previously (see 3.1). Matrix [T] defines the relation between M and B. Each individual element of [T] is listed below.

$$
\begin{aligned}
T_{11} &= \cos D_5 \cos D_6 \\
T_{12} &= \cos D_4 \sin D_5 \\
T_{13} &= -\sin D_5 \\
T_{14} &= 0 \\
T_{21} &= \sin D_4 \sin D_5 \cos D_6 - \cos D_4 \sin D_6 \\
T_{22} &= \sin D_4 \sin D_5 \sin D_6 + \cos D_4 \cos D_6 \\
T_{23} &= \cos D_5 \sin D_4 \\
T_{24} &= 0 \\
T_{31} &= \cos D_4 \sin D_5 \cos D_6 + \sin D_4 \sin D_6 \\
T_{32} &= \cos D_4 \sin D_5 \sin D_6 - \sin D_4 \cos D_6 \\
T_{33} &= \cos D_4 \cos D_5 \\
T_{34} &= 0 \\
T_{41} &= D_1 \\
T_{42} &= D_2 \\
T_{43} &= D_3 \\
T_{44} &= 1
\end{aligned}
$$

Line (3) computes the coordinates of vertex i of the maneuverable plate. Before the transformation, M's position is equal to B's. Since M is rotated 180 degrees relative to B, M's vertex i equals B's vertex p, where p is equal to 1 + (i+2) mod 6 and ($1 \leq p,i \leq 6$) (see section 3.2). Below, the coordinates of M_i after transformation are shown in more detail.

$$\begin{pmatrix} M_{i1} \\ M_{i2} \\ M_{i3} \\ 1 \end{pmatrix} = \begin{pmatrix} T_{11}B_{p1} + T_{21}B_{p2} + T_{31}B_{p3} + D_1 \\ T_{12}B_{p1} + T_{22}B_{p2} + T_{32}B_{p3} + D_2 \\ T_{13}B_{p1} + T_{23}B_{p2} + T_{33}B_{p3} + D_3 \\ 1 \end{pmatrix}$$

Line (3) will compute the length of actuator i. This is equal to the distance between vertex i of the maneuverable and vertex i of the base plate.

$$A_i = \text{sqrt}[\ (M_{i1}-B_{i1})^2 + (M_{i2}-B_{i2})^2 + (M_{i3}-B_{i3})^2\ ]$$

4.2.3. Inverse Transformation

A segment is driven by actuator extensions. When motion is desired by directly changing the length of one or more actuators, an inverse transformation to each degree of freedom is necessary. The objective of this section is to compute the DOF variables D_i ($1 \leq i \leq 6$) given the actuator lengths A_i ($1 \leq i \leq 6$).

One might question whether it is possible to change the length of only one actuator. The fact that a particular set of D-values can be altered in such a way, that only one value of the corresponding set of actuator lengths changes, implies that one or more A-values may be changed and that a corresponding set of D-values exists.

The process of deriving the DOF variables directly from the actuator lengths is rather complex. However, a small change in a segment's actuator lengths, will result in new DOF variable values, close to the current ones. This property can be used by Newton-Raphson approximation to compute these new DOF variables [Dieudonne72].

The Newton-Raphson approximation [Hildebrand74] is an iteration method and will be used to find the common roots of six functions $F_i(D)$ ($1 \leq i \leq 6$). Each function has six parameters, the DOF variables. $F_i(D)$ will be defined as $|B_i-M_i| - A_i$, which represents the distance between vertex i of the maneuverable and base plate, minus the length of actuator i. If the D-parameters are the correct set of values corresponding to the given set of A-values, the functions will be zero for each i ($1 \leq i \leq 6$). The following example is used to derive the general approximation equation.

Given the two functions f(x,y) and g(x,y), find the set (x,y) for which f(x,y) = g(x,y) = 0. The first step is to define f_x as $\partial f(x,y)/\partial x$ and f_y as $\partial f(x,y)/\partial y$ and similarly g_x and g_y. Newton-Raphson method:

$$(x_{k+1}-x_k)\ f_x + (y_{k+1}-y)\ f_y = -f(x_k,y_k)$$

$$(x_{k+1}-x_k)\ g_x + (y_{k+1}-y)\ g_y = -g(x_k,y_k)$$

$$\begin{pmatrix} f_x & f_y \\ g_x & g_y \end{pmatrix} \cdot \begin{pmatrix} x_{k+1}- x_k \\ y_{k+1}- y_k \end{pmatrix} = - \begin{pmatrix} f(x_k, y_k) \\ g(x_k, y_k) \end{pmatrix}$$

$$\begin{pmatrix} x_{k+1}- x_k \\ y_{k+1}- y_k \end{pmatrix} = - \begin{pmatrix} f_x & f_y \\ g_x & g_y \end{pmatrix}^{-1} \cdot \begin{pmatrix} f(x_k, y_k) \\ g(x_k, y_k) \end{pmatrix}$$

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} - \begin{pmatrix} f_x & f_y \\ g_x & g_y \end{pmatrix}^{-1} \cdot \begin{pmatrix} f(x_k, y_k) \\ g(x_k, y_k) \end{pmatrix}$$

The general equation to be used for the inverse actuator extension transformation can be easily derived (see next algorithm line 7). The following algorithm 4.2.3 will compute the DOF variables D given the actuator lengths A.

(1) **REPEAT**

(2) compute [T] and M

(3) $C_i = M_i - B_i$ $(1 \le i \le 6)$

(4) $F_i = |C_i|^2 - |A_i|^2$ $(1 \le i \le 6)$

(5) $J_{pq} = \partial F_p / \partial D_q$ $(1 \le p,q \le 6)$

(6) compute $[J]^{-1}$

(7) $D = D - [J]^{-1} \cdot F(D)$

(8) **UNTIL** F < epsilon.

Algorithm 4.2.3. Before the first iteration at line (1), the values of B and A are assumed to be known. Vector D must be initialized, preferably with values close to the real values which match the given set of actuator lengths A. This way convergence is guaranteed and the amount of necessary iterations minimized.

Line (2) will compute the transformation matrix [T] and the coordinates of the maneuverable plate's vertices M according to algorithm 4.2.2.

Line (3) will compute vector C_i for all i ($1 \le i \le 6$), which represents the distance between vertex i of maneuverable and base plate. It embodies the expected length of actuator i. This expected length is an approximation of the real (known) length of actuator i, represented by A_i. The value of C_i depends on the current value of DOF variables D, which might not yet be the set that matches the A-values. When more loop iterations have been executed, the difference between C and A will become smaller and smaller.

Line (4) computes the value of function F_i for all i ($1 \le i \le 6$). It represents the difference between the expected length of actuator i (according to the current values of DOF variables D) and the real length of actuator i. The advantage of using squared lengths is simplification of the partial derivatives by avoiding complicated and expensive constructions like square roots. When more loop iterations have been executed, the function values will get closer and closer to zero.

Line (5) will compute all 36 values of the 6x6 matrix Jacobian [J], representing the partial derivatives of F by D. Each element D_{ij} is equal to dF_i/dD_j ($1 \le i,j \le 6$) and is shown below.

$$\partial F_i / \partial D_k = 2 \cdot C_{ki} \qquad \text{(all k, } 1 \le k \le 3\text{)}$$

$$\partial F_i / \partial D_4 = 2 \cdot \sum_{1 \le q \le 6} [\, C_{iq} \cdot (T_{3q} B_{i2} - T_{2q} B_{i3}) \,]$$

$$\partial F_i / \partial D_5 = 2 \cdot (C_{i1} \cos D_6 + C_{i2} \sin D_6) \cdot \sum_{1 \le p \le 3} (T_{p3} B_{ip}) - 2 \cdot C_{i3} \cdot (B_{i1} \cos D_5 + \sin D_5 \cdot (B_{i2} \sin D_4 + B_{i3} \cos D_4))$$

$$\partial F_i / \partial D_6 = 2 \cdot (M_{i2} B_{i1} - M_{i1} B_{i2})$$

Line (6). Since the inverse matrix J^{-1} is needed, LU-decomposition [Press86] is used to invert J.

Line (7) where the current DOF values D are adjusted is the heart of the algorithm. For each individual element D_i the approximation can be written as follows.

$$D_i = D_i - \sum_{1 \le p \le 6} (J^{-1}_{ip} \cdot F_p) \qquad \text{(all i, } 1 \le i \le 6\text{)}$$

Line (8) will determine whether the current values of the DOF variables D are satisfactory. This is done by examining the absolute biggest function value F_i ($1 \le i \le 6$). If this value is smaller than a chosen positive number close to zero (epsilon), the DOF values D are accepted. If not, the loop will start its next iteration at line (2).

Convergence is not always guarantied. Some safety precautions must be taken to recognize divergence and avoid a perpetual loop. This can be done by comparing the current function values to their previous ones. Although some fluctuations are allowed, the function values should get smaller during each iteration. Divergence can be caused by initial DOF values which differ too much from the correct corresponding set of D-values that the algorithm tries to find. A second cause for divergence is caused by linear dependency of the elements of the Jacobian. This is the result of an actuator and plate whose angle is close to zero. A subprogram prevents this angle from exceeding a minimum value.

4.3. Indirect Kinematics.

Goal driven motion is achieved by specifying explicitly a desired position and orientation for the control pod. The configuration of the Flexible Arm Robot for the given location will be calculated and expressed in both actuator lengths and DOF variables. Chapter 5 will show how the Arm will be moved automatically from the current to the desired location.

The objective of this section is to derive the DOF variables D from a given location P (P_1,P_2,P_3) of the control pod. These coordinates will be relative to the basis of the Arm, attached to the Habitat Module of the Space Station. An equal-biased control strategy has been adopted. Because the configuration of each segment is the same, the stacked segments of the Flexible Arm Robot will form an arc, from the Habitat Module to the control pod. The index u will be omitted since the values of the six-tuples A_u (as well as D_u) are equal for each segment u ($1 \le u \le n$).

4.3.1. List of Variables

P Location of control pod.

(P_1, P_2, P_3)

L Orthonormal projection of P on the XOY-plane.

$(L_1, L_2, L_3) = (P_1, P_2, 0)$

M Center of circle through O and P.

$(M_1, M_2, M_3) = (M_1 , M_2, 0)$

U Location maneuverable plate of segment 1.

$(U_1, U_2, U_3) = (D_1 , D_2 , D_3)$

T Orthonormal projection of U on the XOY-plane.

$(T_1, T_2, T_3) = (U_1 , U_2, 0)$

S Location maneuverable plate of segment 1 without x,y-rotations.

$(S_1, S_2, S_3) = (S_1 , S_2, 0)$

Q Location maneuverable plate of segment 1 without y-rotation.

(Q_1, Q_2, Q_3)

O Origin of coordinate axes (0,0,0), basis of FAR.

E Center of circle through O and Q.

k Ratio M/L.

r Radius of circle through O and P.

R_w Angle of last segment and the XOY-plane.

R_o Angle of first segment and the XOY-plane.

R_x Angle between line OS and line OQ

R_y Angle between line OQ and line OU.

4.3.2. Configuration Transformation

Figures k1 to k4 will illustrate the transformation from pod coordinates P to DOF variables D. Finally, the actuator lengths A can be easily derived from D (see section 4.2.2).

The configuration of the Flexible Arm Robot in figure k1 is represented by the arc from origin O to P. Point P is the location of the control pod. Origin O is the center of the base plate of segment one, which is attached to the Habitat Module. The arc is part of the circle through O and P with midpoint M and radius r. It contains all centers of the Arm's plates and is situated in the shaded rectangle. Point U represents the center of segment one's maneuverable plate. The angle between this plate and the XOY-plane is represented by R_o. R_w is total angle of the Arm, from the base to the control pod. Point L is the orthonormal projection of P on the XOY-plane, and T the orthonormal projection of U. The shaded triangle OTU can also be found in figure k3. A straight line through O and M cuts T and L (not necessarily in that order). Obviously, midpoint M can be expressed in terms of L. Since L is equal to $(P_1,P_2,0)$, M can be expressed in L times a constant factor k $(kP_1,kP_2,0)$. If R_w is equal to 90 degrees, than L is equal to M, and k = 1. Thus, if $k < 1$ then $R_w < 90$ degrees, if $k > 1$ then $90 < R_w < 180$ degrees.

First k can be expressed in P, since $|OM| = |PM|$

$$(kP_1)^2 + (kP_2)^2 = (P_1 - kP_1)^2 + (P_2 - kP_2)^2 + P_3^2$$

$$0 = P_1^2 - 2kP_1^2 + P_2^2 - 2kP_2^2 + P_3^2$$

$$k = \frac{1}{2}\left(1 + \frac{P_3^2}{P_1^2 + P_2^2}\right)$$

Now r can be expressed in P, since $r = |OM|$.

$$r = k \cdot \mathrm{sqrt}(P_1^2 + P_2^2)$$

Next, $\sin(R_w) = |PL|/r$, while $|PL| = P_3$, so that

$$R_w = \arcsin\left(\frac{P_3}{r}\right)$$

$$R_o = R_w / n$$

The first DOF variable to be determined is DOF variable D_3, the z-translation of segment one. Since U is the center of the maneuverable plate of segment one, D_3 is equal to |UT| which is equal to r . $\sin R_o$.

$$D_3 = r \cdot \sin R_o$$

Finally, the length of line OT will be determined. Clearly, |OT| is equal to |OM| - |TM|. But |OM| = r and |TM| = r . $\cos R_o$, so that

$$|OT| = r - r \cdot \cos R_o$$

$$= r \cdot (1 - \cos R_o)$$

Figure k2 is an orthonormal view of figure k1. It shows the XOY-plane, with the z-axis pointing upward from the paper toward the viewer's eye. Figure k2 will assist determining the x and y translations of segment one, represented by DOF variables D_1 and D_2. D_1 is equal to U_1 and D_2 equal to U_2. The relation $U_1/|OT| = kP_1/|OM|$ and $U_2/|OT| = kP_2/|OM|$ is obvious. |OM| is equal to radius r.

$$U_1 / |OT| = kP_1 / |OM|$$

$$U_1 = k \cdot P_1 \cdot |OT| / |OM|$$

$$D_1 = k \cdot P_1 \cdot (1 - \cos R_o)$$

$$D_2 = k \cdot P_2 \cdot (1 - \cos R_o)$$

Figure k3 shows a sphere with center O. Points U, Q and S are located on the surface of the sphere. Furthermore, S is positioned on the z-axis, Q is part of the YOZ-plane and |OU| = |OS| = |OQ| is the radius of the sphere. Lines SU, SQ and QU are also part of the sphere. The triangle OTU can also be found in figure k1. U is the center of the maneuverable plate of segment one. Point Q would be this location if the y-rotation D_5 was ignored. Point S would be this location if both x and y rotation D_4 and D_5 were omitted. R_x is the angle between OS and OU. R_y is the angle between OQ and OU.

After point S has been rotated R_x degrees around the x-axis, it is equal to point Q. After point Q has been rotated R_y degrees around the y-axis, it is equal to point U. This results in the following relations.

$$\begin{pmatrix} 0 \\ 0 \\ |OU| \end{pmatrix}^T \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos R_x & \sin R_x \\ 0 & \sin R_x & \cos R_x \end{pmatrix} = \begin{pmatrix} 0 \\ |OU|\ \sin R_x \\ |OU|\ \cos R_x \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ |OU|\ \sin R_x \\ |OU|\ \cos R_x \end{pmatrix}^T \cdot \begin{pmatrix} \cos R_y & 0 & -\sin R_y \\ 0 & 1 & 0 \\ \sin R_y & 0 & \cos R_y \end{pmatrix} = \begin{pmatrix} -|OU|\ \cos R_x \sin R_y \\ |OU|\ \sin R_x \\ |OU|\ \cos R_x \cos R_y \end{pmatrix}$$

This last vector is equal to U (U_1,U_2,U_3), so that both R_x and R_y can be expressed in U_i, which is equal to D_i $(1 \le i \le 3)$.

$$|OU|^2 = U_1^2 + U_2^2 + U_3^2$$

$$R_x = \arcsin(U_2 / |OU|)$$

$$R_y = \arcsin(-U_1 / |OU| \cos R_x)$$

Figure k4 shows the relation between R_x and D_4. The YOZ-plane is shown containing a circle through O and Q with center E and radius $|OE| = |OQ|$. The shaded isosceles triangle OEQ can also be found in figure k3. Angle R_x is known, but angle D_4 is required for this is the actual x-rotation of segment one. The figure shows that D_4 equals $2R_x$. Similarly, the y-rotation D_5 is equal to $2R_y$. The z-rotation D_6 may keep its initial value.

$$D_4 = 2 \cdot R_x$$

$$D_5 = 2 \cdot R_y$$

5. MOVEMENT

The direct and indirect kinematics in the previous chapter are used to create control algorithms for a moving Flexible Arm Robot. These algorithms can be applied to both the computer-simulated and the real Flexible Robot. Two different methods will be examined; data driven and goal driven motion.

5.1. Data Driven Motion

Data driven motion is a technique in which the Flexible Robot is moved by directly manipulating the parameters which define its configuration. The configuration parameters to be manipulated are the DOF variables and actuator lengths. This type of movement can also be referred to as 'manual controlled' motion. The computer system that regulates the motion of the Arm will receive signals from input devices, such as keyboard, joy stick, mouse, dial box, etc. These signals are translated into modifications of one or more configuration parameters. The computer simulation visualizes motion by changing these parameters during the consecutive drawings of the Flexible Arm Robot on the computer screen. The speed at which the object moves is determined by the (possibly negative) value, which is added to the current value of the parameter. A higher increment will make the object move faster.

```
(1)  LOOP
(2)       ClearScreen
(3)       Draw_Flexible_Arm_Robot( D )            (algorithm 3.3)
(4)       IF  Change_DOFvariable  THEN
(5)            FOR  i=1  TO  6  DO
(6)                 IF  change( i )  THEN   D_i  =  D_i  +  delta()   ENDIF
(7)            ENDDO
(8)            A  =  DOF_to_Actuator( D )         (algorithm 4.2.2)
(9)       ELSEIF  Change_ActuatorLength  THEN
(10)           FOR  i=1  TO  6  DO
(11)                IF  change( i )  THEN   A_i  =  A_i  +  delta()   ENDIF
(12)           ENDDO
(13)           D  =  Actuator_to_DOF( A )         (algorithm 4.2.3)
(14)      ENDIF
(15) ENDLOOP
```

Algorithm 5.1 uses the direct kinematics of sections 4.2. The statement sequence from line (1) to (15) is a perpetual loop. Each time after the computer screen has been cleared (line 2), the Flexible Arm Robot is drawn (line 3), using the current values of the DOF variables D.

After line (3), either the DOF variables D or the actuator lengths A can be changed. If the boolean value Change_DOFvariables is true, the statement sequence line (5) to (8) will be executed. If not, and the boolean value Change_ActuatorLength is true, the sequence line (10) to (13) will be executed.

The loop from line (5) to (7) will give each DOF variable D_i ($1 \leq i \leq 6$) a chance to be changed.

Line (6) will add the result of function delta(), the parameter-increment, to DOF variable D_i if the boolean function change(i) returns the value 'true'. In this simulation, change(i) is triggered by one of the three buttons of a 'mouse' input-device, in combination with a 'r' (for rotation) or 't' (for translation) keystroke on the keyboard. Function delta() is used to control the speed of the movement. Its value is set by the xy-value of the same mouse device.

Line (8) will compute the new actuator lengths A, which corresponds to the changed D value(s), using algorithm 4.2.2.

Line (9) to (13) will change the actuator lengths as line (4) to (8) changed the DOF variables. Consequently, line (13) will compute the DOF variables D, corresponding to the new actuator lengths A, using algorithm 4.2.3.

5.2. Goal Driven Motion

Goal driven motion is a technique in which the Flexible Arm Robot is moved automatically from its current location to a specified location. The configuration of the Arm in this position is calculated using the inverse kinematics of section 4.3, and expressed in terms of DOF variables and actuator lengths. A so called nonlinear maneuvering algorithm has been developed to move the Flexible Robot from its current to its desired configuration. The algorithm will give the DOF variables or actuator lengths their desired values after a fixed number of steps. Each step, the DOF variables or actuator lengths will be altered and close in on their goal values. The total amount of steps required, depends on the difference between each actuator's or DOF variable's current and goal value. The biggest difference determines the required number of steps, for this parameter will experience the highest increment per step. If the increment is too great, the motion of the object will not be smooth. Each DOF variable (translation, rotation) and actuator length has an empirically determined maximum allowable increment value.

5.2.1. Nonlinear Maneuvering based on DOF Variables

After the desired location of the control pod is given, the corresponding goal-DOF-variables G_i ($1 \le i \le 6$) are computed. When the number of steps s in which the Arm will be moved to its goal location is defined, the increment per DOF variable D_i per step is computed. This constant value C_i will be added to each DOF variable D_i ($1 \le i \le 6$) during each step, while the Flexible Arm Robot is drawn on the computer screen. The last step will give each DOF variable its desired value.

```
(1)     FOR i=1 TO 6 DO
(2)         C_i = G_i - D_i
(3)     ENDDO
(4)     s = MaxFunc( C_1 , C_2 , C_3 , C_4 , C_5 , C_6 )
(5)     FOR i=1 TO 6 DO
(6)         C_i = C_i / s
(7)     ENDDO
(8)     WHILE s>0 DO
(9)         s = s - 1
(10)        FOR i=1 TO 6 DO
(11)            D_i = G_i - s * C_i
(12)        ENDDO
(13)        A = DOF_to_Actuator( D )            (algorithm 4.2.2)
(14)        Draw_Flexible_Arm_Robot( D )        (algorithm 4.3.2)
(15)    ENDDO
```

Algorithm 5.2.1. Line (2) assigns to C the difference between goal value G and current value D. This value may be negative.

Line (4) determines s, the required number of steps. Its (integer) value depends on the absolute maximum value of C_i ($1 \le i \le 6$). The function MaxFunc will take into consideration that the first three parameter values to be compared to the others represent translations, and the last three values rotations.

Line (6) gives each C_i ($1 \le i \le 6$) its intended value, the increment per DOF variable D_i per step, by dividing each C_i by s.

The while-loop from line (8) to (15) will handle the actual movement. For large values of s, the computer's internal precision may cause the final DOF values not to be exactly equivalent to the desired values G. In order to end the while-loop with the correct DOF values, the iteration strategy is inverted. Instead of adding a value during each iteration, a value will be subtracted.

Variable s is the number of remaining steps. Line (9) will subtract one during each iteration. Line (11) computes the new current value of the DOF variables D. This is done by subtracting s times the difference per step from the goal value G. Before the first iteration, the assignment is D = G - (G-D) which is equal to D. Thus, the loop will start with the correct value of D. The last iteration, s is equal to zero. The assignment is now D = G - 0, so that D will get the exact desired value G. Finally, line (13) derives the actuator lengths A from the DOF variables D.

5.2.2. Nonlinear Maneuvering based on Actuator Lengths

Although the concept of a nonlinear maneuvering algorithm based on the actuator lengths is not much different from that based on DOF variables, the behavior of the moving Flexible Arm Robot is significantly better (see section 5.2.3). Its major disadvantage is the relatively high-cost conversion of actuator lengths to DOF variables (see line 13 below).

```
(1)     FOR i=1 TO 6 DO
(2)         C_i = G_i - A_i
(3)     ENDDO
(4)     s = MaxFunc( C_1 , C_2 , C_3 , C_4 , C_5 , C_6 )
(5)     FOR i=1 TO 6 DO
(6)         C_i = C_i / s
(7)     ENDDO
(8)     WHILE s>0 DO
(9)         s = s - 1
(10)        FOR i=1 TO 6 DO
(11)            A_i = G_i - s * C_i
(12)        ENDDO
(13)        D = Actuator_to_DOF( A )            (algorithm 4.2.3)
(14)        Draw_Flexible_Arm_Robot( D )        (algorithm 4.3.2)
(15)    ENDDO
```

Algorithm 5.2.2. This time, after the desired location of the control pod is given, the corresponding goal-Actuator-lengths G_i ($1 \le i \le 6$) are computed using the indirect kinematics of section 4.3 and algorithm 4.2.2. The constant value C_i is now the difference between the current length and goal length of actuator i ($1 \le i \le 6$), and will be added during each loop iteration, while the Flexible Arm Robot is drawn on the computer screen. The last step will give each actuator its desired length.

Only lines different from algorithm 5.2.1. will be discussed. At line (4), function MaxFunc can determine the number of steps s more easily since all constant values C_i ($1 \le i \le 6$) are of the same type. Line (11) will now compute the new current actuator lengths A. Line (13) computes the corresponding DOF variables D, using algorithm 4.2.3.

5.2.3. Characteristics of Nonlinear Maneuvering Algorithm.

The advantage of the algorithm based on actuator lengths involves the behavior of the actuators during movement. One example would be if the Flexible Arm Robot has to be moved from a -10 degree to a 20 degree x-rotation. This can be done in 10 steps, by adding 3 degrees to the current x-rotation during each step. The Arm will move to its desired configuration in a smooth way. However, examination of the actuator lengths during each step shows, that some of them increase their lengths after initially getting shorter. This results in a serious waste of energy for a fifty segment Flexible Arm Robot, which has three hundred actuators. Furthermore, if a segment is in an extreme position, motion may be impossible if based on nonlinear maneuvering based on the DOF variables, as illustrated in figure m2. If nonlinear maneuvering is based on the actuator lengths, the Flexible Arm Robot will experience no problems moving from the -10 degree to the 20 degree x-rotation and problems with segments in extreme positions no longer exist. Each actuator whose length has to be changed, will extend or retract, step by step, to its desired length. In contrast to nonlinear DOF maneuvering, actuators whose initial and goal length are the same, will not change during movement.

The segment in figure m1 is in balanced position The lengths of actuators A1, A2 and A3 are the same, and equal to (minlength + maxlength)/2. Joint J1 connects actuators A1 and A2 to the maneuverable plate, and joint J2 connects actuator A3 to the maneuverable plate. Parts of four circles with their intersections E, F, G and H are shown. These arcs represent the extreme values of the three actuators. The arc through E and H is the minimum length of actuators A2 and A3. The arc through G and F represents the maximum values of actuators A2 and A3. The arc through E and F is the minimum, the arc through G and F the maximum length of actuator A1. Thus, joint J1 must be inside the area enclosed by E, F, G and H. If this joint is located in one of these points, and actuator A3 is either minimal or maximal, the segment is in an extreme position.

In figure m2, the same segment as in figure m1 in a different position. The segment is in an extreme position. Joint J1 is located in point F and actuator A3 is maximal. The dotted lines illustrate the same segment, with joint J1 moved from point F to point E. Again the segment is in an extreme position, for the lengths of actuators A1 and A2 are now minimal. One arrow shows the trajectory of the center of the maneuverable plate, when the nonlinear maneuvering is based on the DOF variables. The other arrow shows the path of joint J1. Clearly, actuator A1 gets initially shorter, and during motion becomes longer. Because its length would exceed its minimum value, the suggested motion is not possible. The figure shows a situation where the nonlinear maneuvering algorithm based on DOF variables fails.

Figure m3 assumes the same situation as figure m2 with the exception the use of nonlinear maneuvering based on actuator lengths. The figure shows the behavior of the maneuverable plate, while the segment is moved from one extreme position in the other. This is done by changing the length of actuator A2, step by step, from its maximum to its minimum value. During this motion, the length of actuators A1 and A3 will not change. Joint J1 will follow the arc from point F to E. Since the length of the (maneuverable) plate is fixed, joint J2 can only move back and forth. The shaded little circles show the trajectory of the center of the maneuverable plate. This figure shows the superiority of the nonlinear maneuvering algorithm based on actuator lengths.

5.2.4. Motion Constraints.

Motion in each degree of freedom can never exceed physical limitations on position. A segment is considered in its balanced position if all the actuators are half extended, that is half way their fully extended and retracted lengths. For each degree of freedom there exists a set of values of the other five degrees of freedom which will allow the maximum motion in that degree of freedom (see table t1). This is called the neutral position of the degree of freedom, which may be different from the balanced position of the segment. Only in neutral position, a degree of freedom can reach its absolute maximum plus or minus value. Because the actuators may be extended only 81.021 cm (31.898 in.), a displacement in one degree of freedom changes the maximum positions that may be obtained individually in each of the other five degrees of freedom (see figure t2). The closer one degree of freedom approaches its absolute plus or minus maximum value, the more the maximum positions in the other five degrees of freedom are limited. [Parrish73] presents a method to predict these position limits, based on the current orientation, of a six-degree-of-freedom flight simulator platform. The results of tables t1 and t2 have been determined empirically.

6. CONCLUSIONS

The powerful concept of a Flexible Arm Robot allows an astronaut to reach any location around the space station without extra vehicular activities. This eliminates the necessity of wearing a space suit. FAR's many degrees of freedom, and consequently great flexibility, required development of more complex control algorithms. The kinematic solutions, motion control algorithms, and its implementation in a graphical computer simulation are presented in this work. The direct kinematic algorithms offer a method for driving the Flexible Arm Robot's position and orientation from the lengths of its actuators, and visa versa. The indirect kinematic algorithms offer a method to derive a configuration for the Flexible Arm Robot for a given location of the control pod. A nonlinear goal-driven maneuvering algorithm, which uses these kinematic equations to move the Flexible Arm Robot, has been developed. The validity of the algorithms has been proven by its implementation in the computer simulation.

The simulation makes it possible to examine the dynamic properties of the Flexible Arm Robot. The simulation has shown that e.g. an equal-biased strategy restricts the flexibility of FAR. It appears that this strategy does not allow a smooth motion of a fully extended configuration. Study of the dynamic characteristics of the Flexible Arm Robot will be an issue for further research.

ORIGINAL PAGE
COLOR PHOTOGRAPH

7. ILLUSTRATIONS

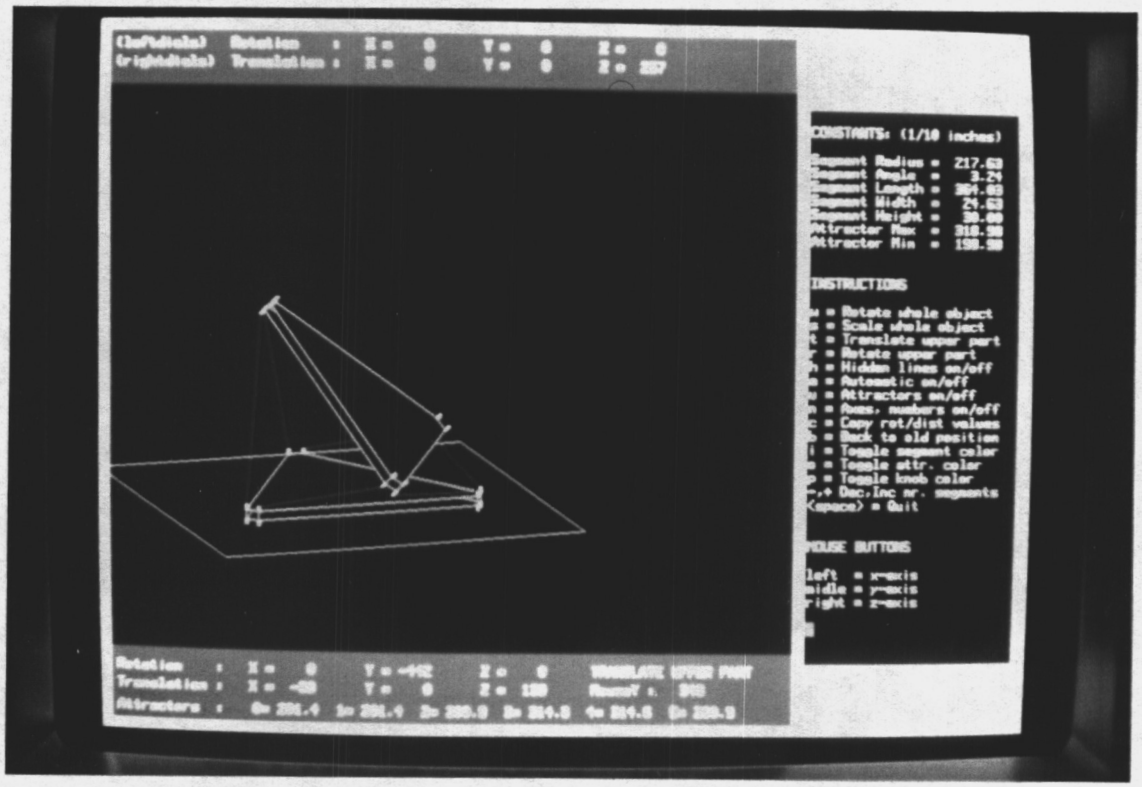


Illustration 1

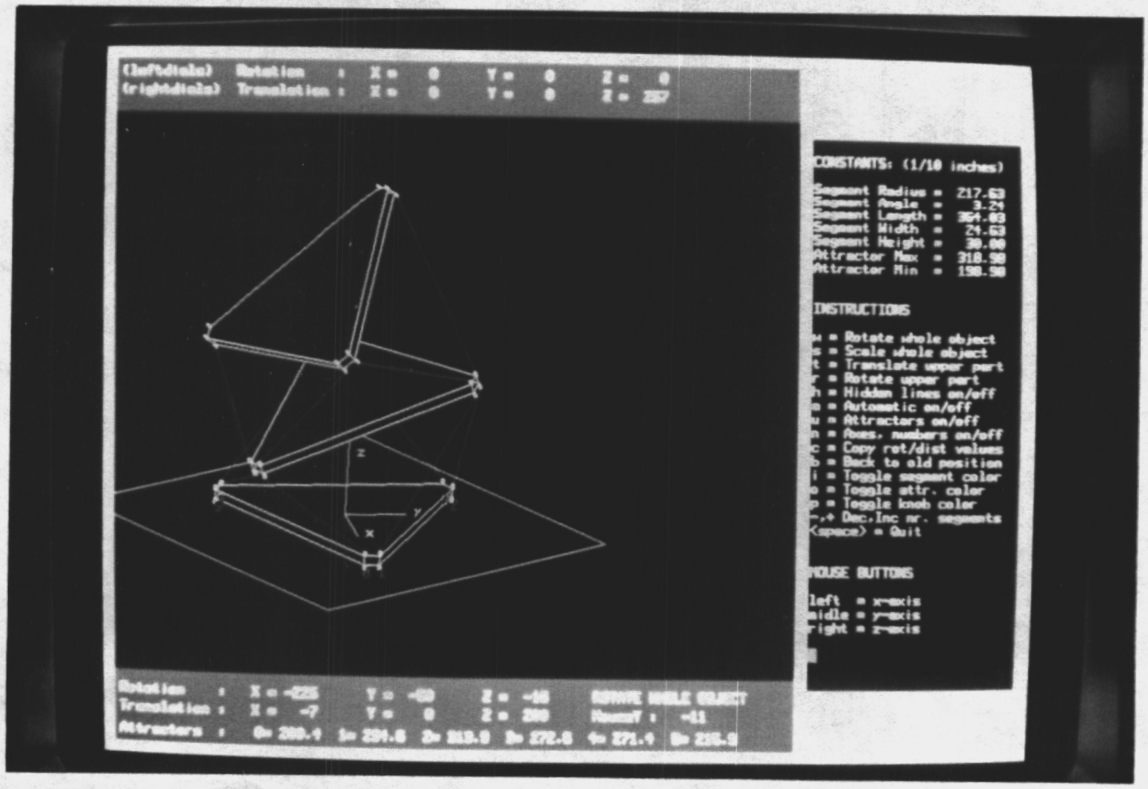


Illustration 2

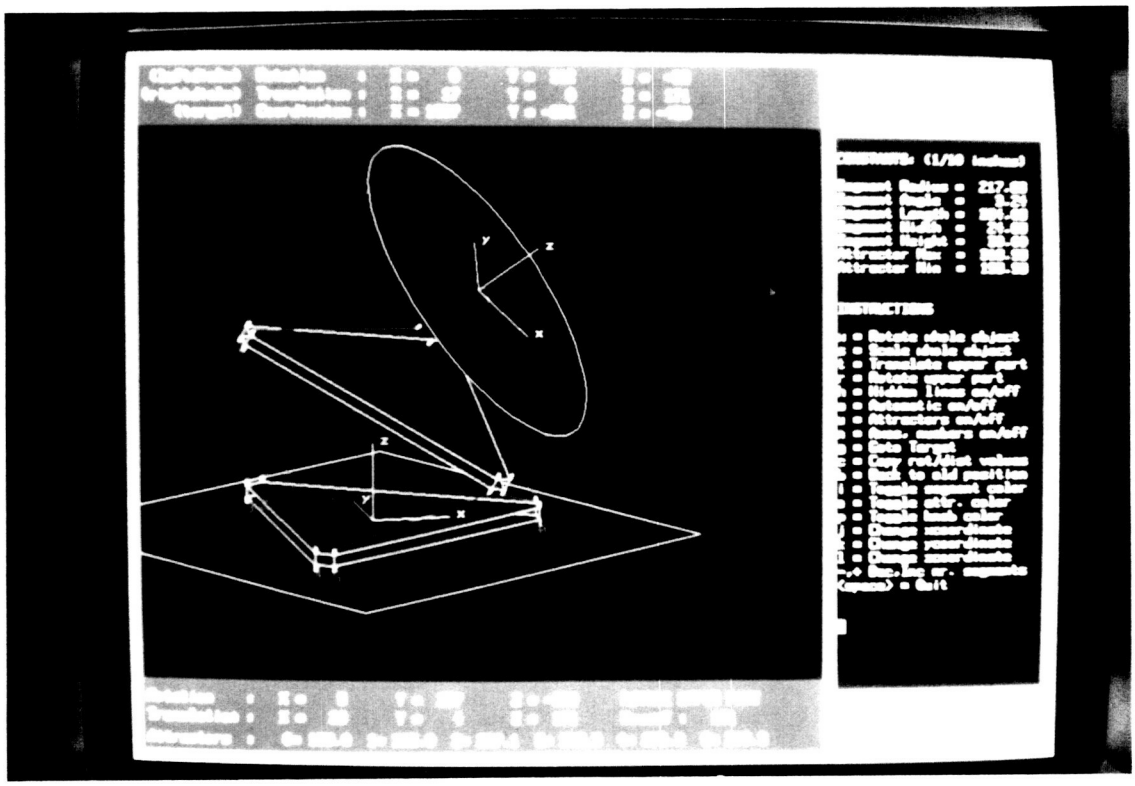


Illustration 3

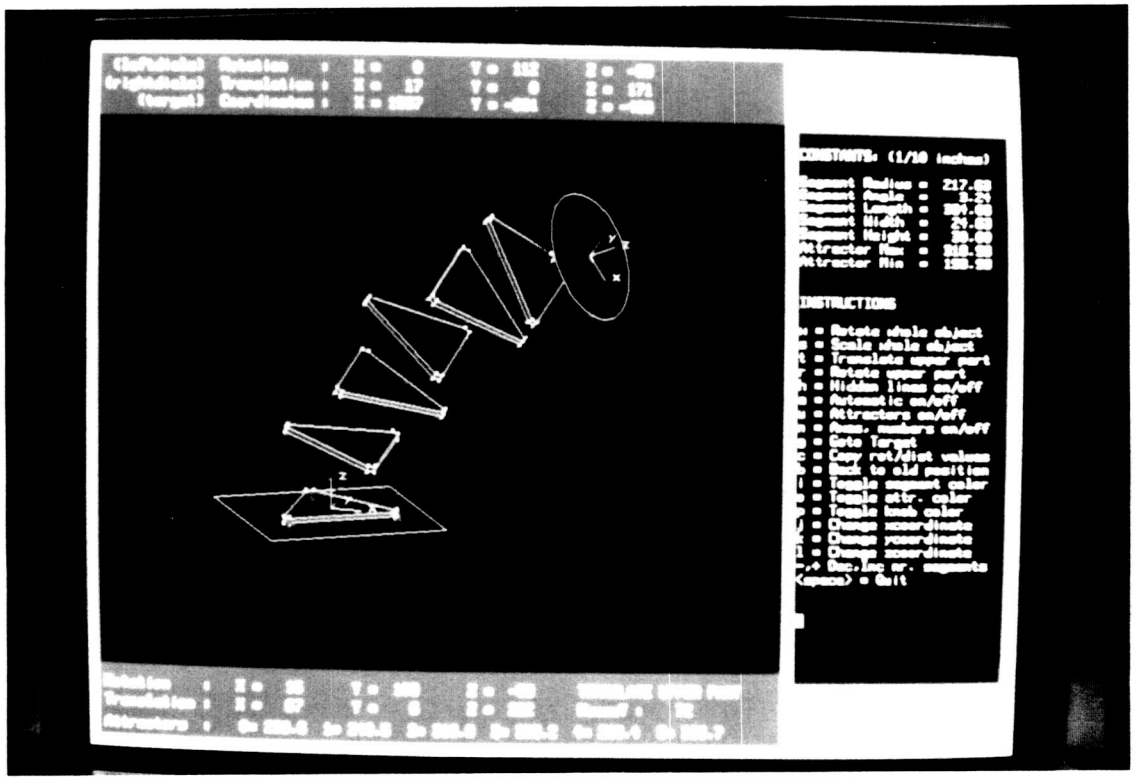


Illustration 4

ORIGINAL PAGE
COLOR PHOTOGRAPH

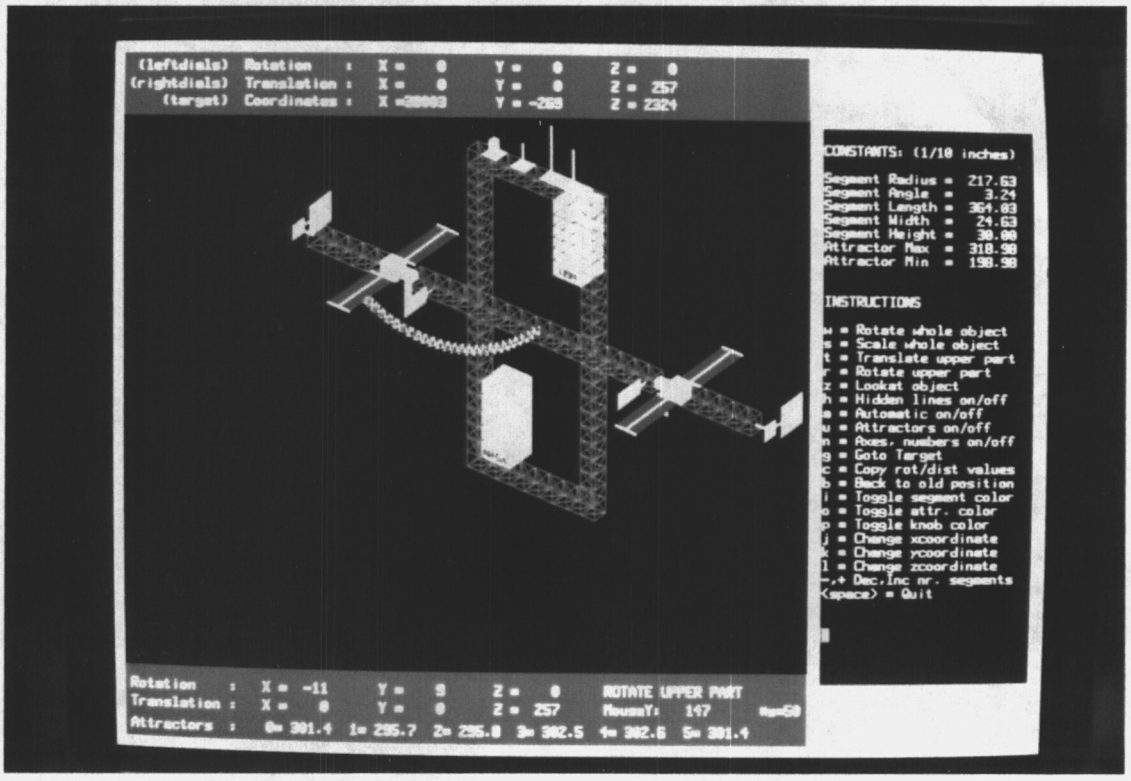


Illustration 5

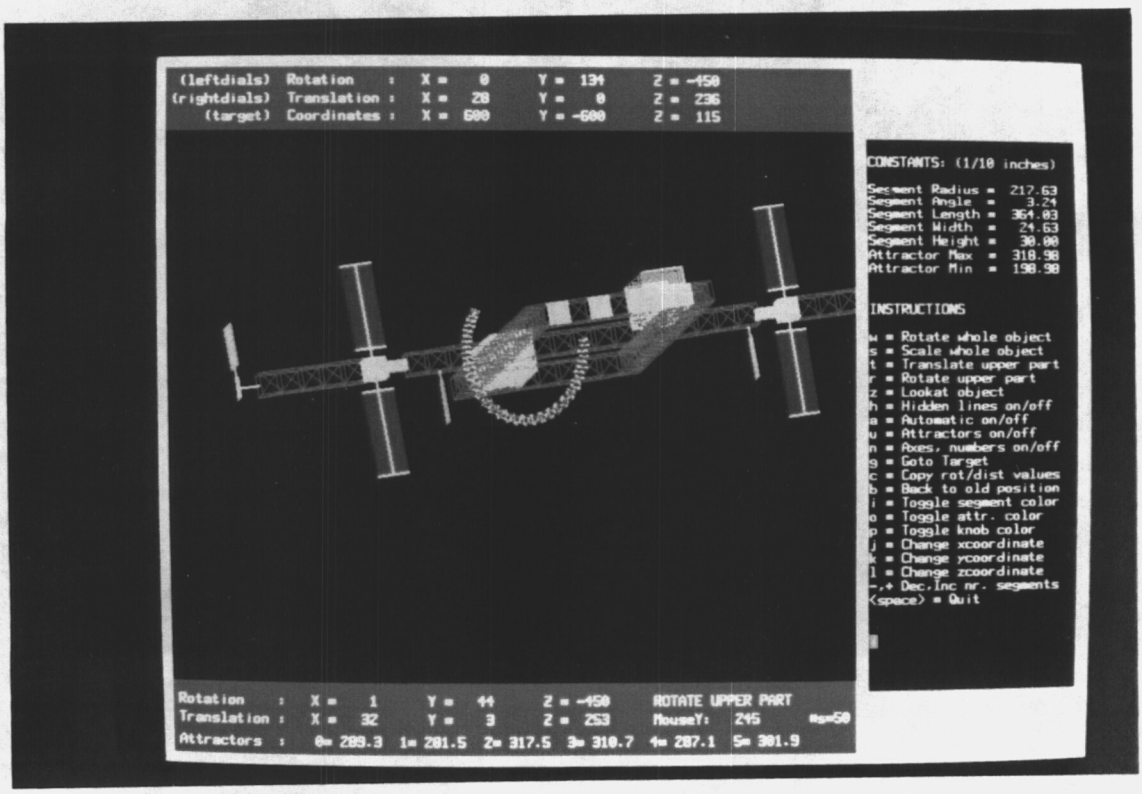


Illustration 6

Index to Illustrations

Illustration 1.

This picture shows the graphical representation of a segment. The yellow hexahedrons represent the base and maneuverable plates. The red lines represent the actuators. The white dots are the joints. The green square symbolizes the Habitat Module of the Space Station. The angle between base and maneuverable plate is approximately 44 degrees.

Illustration 2

Two segments. The base plate of segment one shows the xyz-axes and vertex numbers. The configuration of both segments is the same (equal-biased strategy).

Illustration 3.

The green circle represents the control pod and shows the local location and orientation of the xyz-axes.

Illustration 4.

This is a six-segment Flexible Arm Robot. Each segment has a y- rotation of approximately 10 degrees, and a translation in the x direction of 17.0 cm (6.7 inch).

Illustration 5.

Space Station with a fifty segment Flexible Arm Robot. The control pod is located at the left solar panel. The angle between the maneuverable and base plate of each segment not more than 1.1 degree.

Illustration 6.

This is a top-view of the Space Station. Again, the Flexible Arm Robot consists of fifty segments. With a y-rotation of only 4.4 degrees, the control pod is able to reach the other side of the Space Station.

8. REFERENCES and WORKS CONSULTED

[Asano84] Asano K.,
Control System for a Multi-Joint Inspection Robot,
Proc. 1984 National Topical Meeting on Robotics and Remote Handling in Hostile Environments, p375-382.

[Dieudonne72] Dieudonne J.E., Parrish R.V., Bardusch R.E.,
An Actuator Extension Transformation for a Motion Simulator,
1972, NASA Langley Research Center, Hampton, VA.

[Fichter] Fichter E.F., McDowell E.D.,
A Novel Design for a Robot Arm,
Oregon State University, Oregon.

[Hildebrand74] Hildebrand F.B.,
Introduction to Numerical Analysis,
1974, McGraw-Hill, Inc. New York.

[Hunt83] Hunt K.H.,
Structural Kinematics of Inparallel-Actuated Robot-Arms,
Journal of Mech., Transm., and Autom. in Design, p705-12 vol 105, December 1983

[Mohamed85] Mohamed M.G., Duffy J.,
A Direct Determination of the Instantaneous Kinematics of Fully Parallel Robot Manipulators,
Transactions of the ASME, p226-9 vol 107, June 1985.

[Newman79] Newman, W.M., Sproull R.F.,
Principles of Interactive Computer Graphics,
1979, Second Edition, McGraw-Hill, Inc. Tokyo, Japan.

[Parrish73] Parrish R.V., Dieudonne J.E., Martin D.J. Jr.,
Motion Software for a Synergistic Six-Degree-Of-Freedom Motion Base,
1973, NASA Langley Research Center, Hampton, VA.

[Press86] Press A.H., Flannery B.P., Teukolsky S.A., Vetterling W.T.,
Numerical Recipes, The Art of Scientific Computing,
1986, Cambrige University Press.

[Stewart65] Stewart D.,
A Platform with Six Degrees of Freedom,
Proc. Instn. Mech. Engrs. (London), 1965-66
Vol. 180 Pt. 1 No. 15.

[Wessling86] Wessling F.C., Bakken G.B., Teoh W., Ziemke M.C., Patel V.,
Personnel Occupied Woven Envelope Robot,
November 30, 1986, Progress Report, Johnson Research Center, Wyle Laboratories, Huntsville, AL.

[Wessling87a] Wessling, F.C.,
Personnel Occupied Woven Envelope Robot,
June 1, 1987, Status Report,
Johnson Research Center, Huntsville AL.

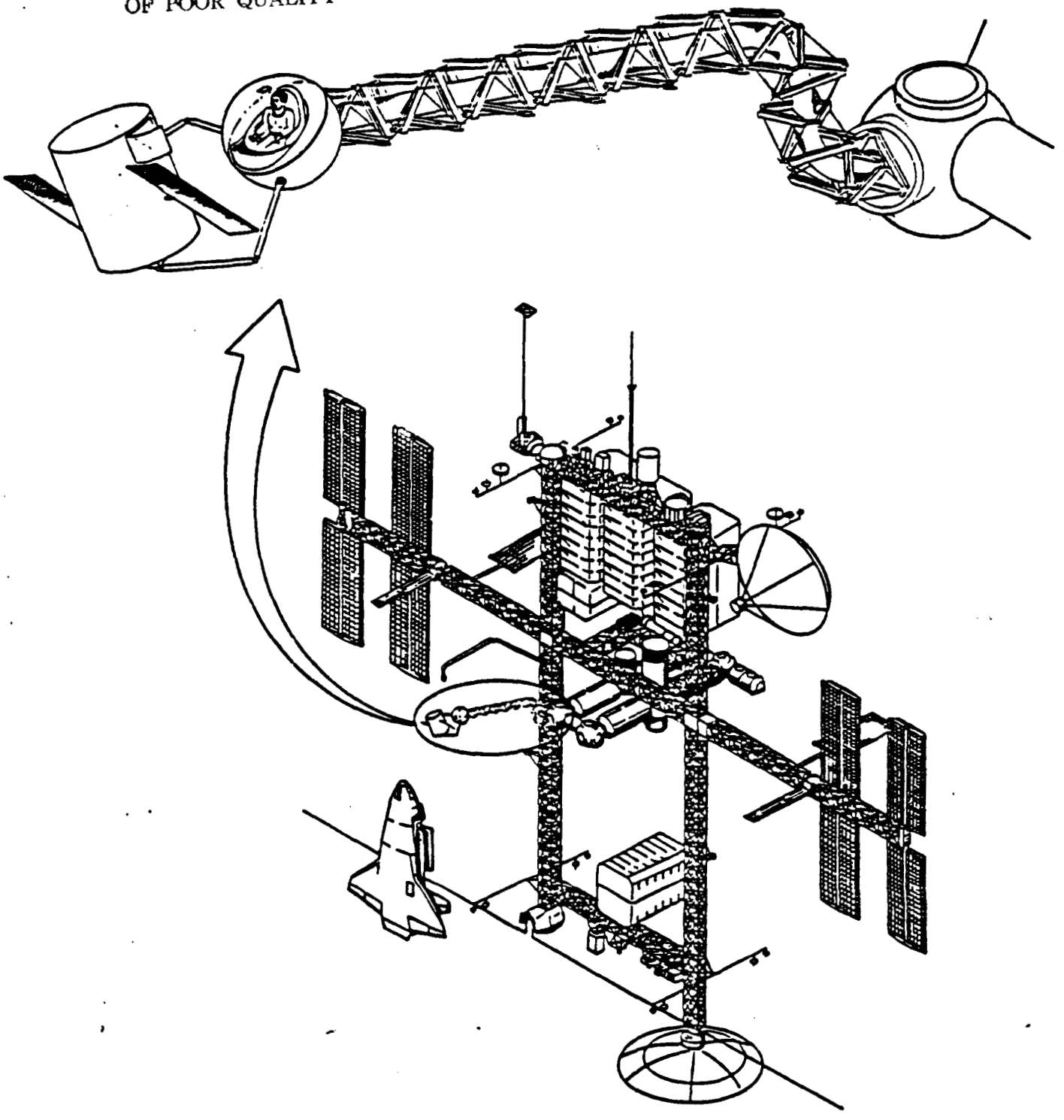
[Wessling87b] Wessling, F.C.,
Personnel Occupied Woven Envelope Robot,
November 30, 1987, Status Report,
Johnson Research Center, Huntsville AL.

DOF-variable	Maximum	Whole set D_{1-6}
D_1 cm (in.)	26.7 (10.5)	(26.7, 0, 52.6, 0, 0, 0)
	-26.7 (-10.5)	(-26.7, 0, 34.0, 0, 0, 0)
D_2 cm (in.)	23.1 (9.1)	(0, 23.1, 46.7, 0, 0, 0)
	-23.1 (-9.1)	(0,-23.1, 46.7, 0, 0, 0)
D_3 cm (in.)	71.4 (28.1)	(0, 0, 71.4, 0, 0, 0)
	16.5 (6.5)	(0, 0, 6.5, 0, 0, 0)
D_4 deg.	33.9	(0, 11.9, 41.9, 33.9, 0, 2.1)
	-33.9	(0,-11.9, 41.9,-33.9, 0,-2.1)
D_5 deg.	49.6	(11.9, 0, 32.3, 0, 49.6, 0)
	-49.5	(-17.0, 0, 30.5, 0,-49.5, 0)
D_6 deg.	23.9	(0, 0, 49.8, 0, 0, 23.9)
	-23.9	(0, 0, 49.8, 0, 0,-23.9)

Table t1 - Absolute plus/minus maxima of DOF variables.

Static Position	Position Limits				
D_1 cm (in.)	D_2 cm (in.)	D_3 cm (in.)	D_4 deg.	D_5 deg.	D_6 deg.
19.1 (7.5)	±13.2 (5.2)	60.2 (23.7) 33.5 (13.2)	±9.0	11.5 -31.8	±5.8
12.7 (5.0)	±23.1 (9.1)	64.8 (25.5) 31.2 (12.3)	±19.0	19.2 -37.5	±11.2
6.4 (2.5)	±22.9 (9.0)	68.3 (26.9) 26.4 (10.4)	±20.5	26.0 -39.7	±17.4
0.0 (0.0)	±23.1 (9.1)	71.4 (28.1) 16.5 (6.5)	±33.9	29.3 -31.1	±23.9
-6.4 (2.5)	±22.9 (9.0)	68.6 (27.0) 25.1 (9.9)	±22.2	31.0 -23.1	±18.1
-12.7 (5.0)	±22.9 (9.0)	64.8 (25.5) 29.0 (11.4)	±18.7	34.6 -15.5	±12.3
-19.1 (7.5)	±13.2 (5.2)	60.2 (23.7) 32.5 (12.8)	±9.4	44.1 -8.4	±6.7

Table t2 - Plus and Minus maximum values of the other five DOF variables for a fixed value of the x-translation D_1.



The University Of Alabama In Huntsville

WYLE LABORATORIES

P.O.W.E.R.

(PERSONNEL OCCUPIED WOVEN ENVELOPE ROBOT)

FIGURE P1 SERVICING A SATELLITE

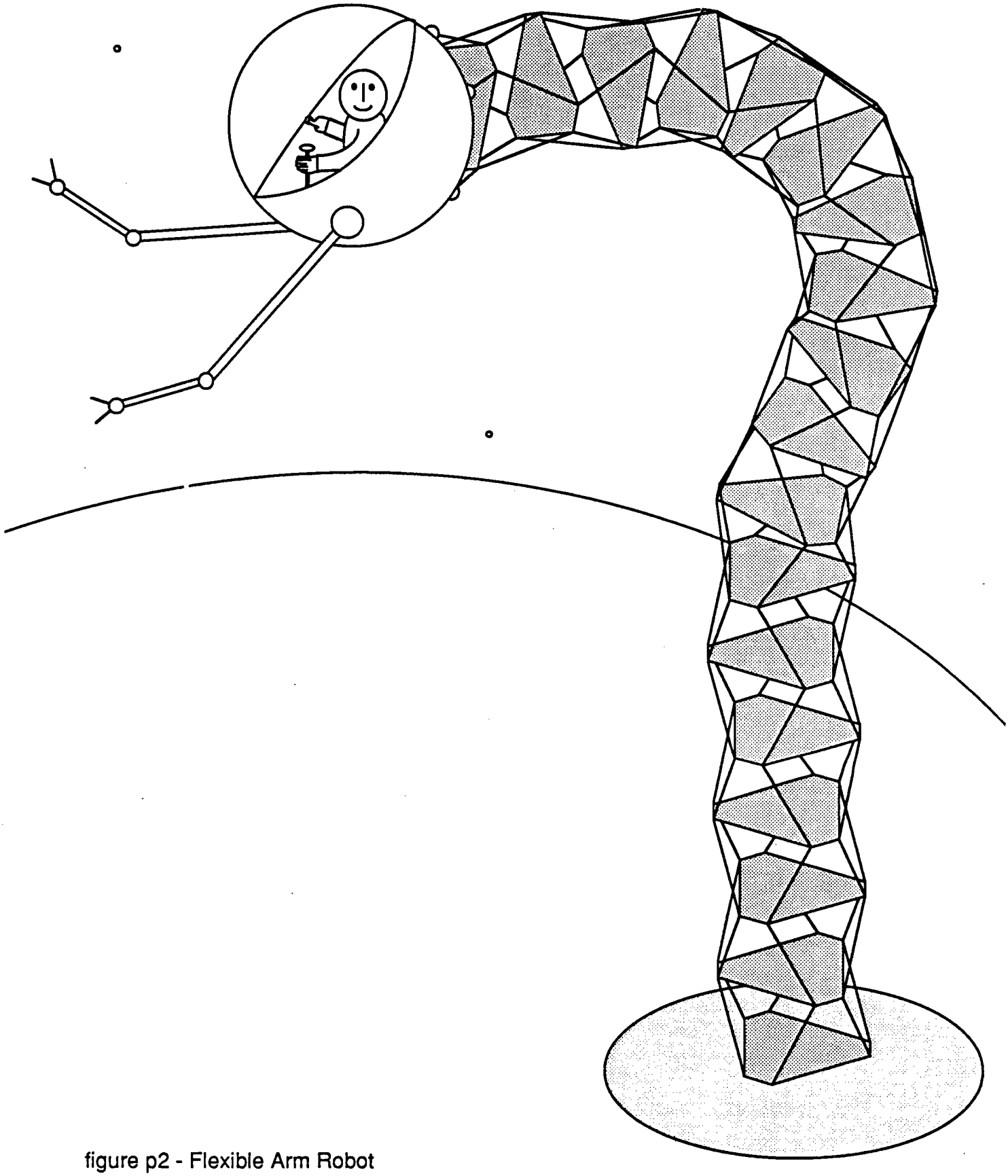


figure p2 - Flexible Arm Robot

PJ-880610-003

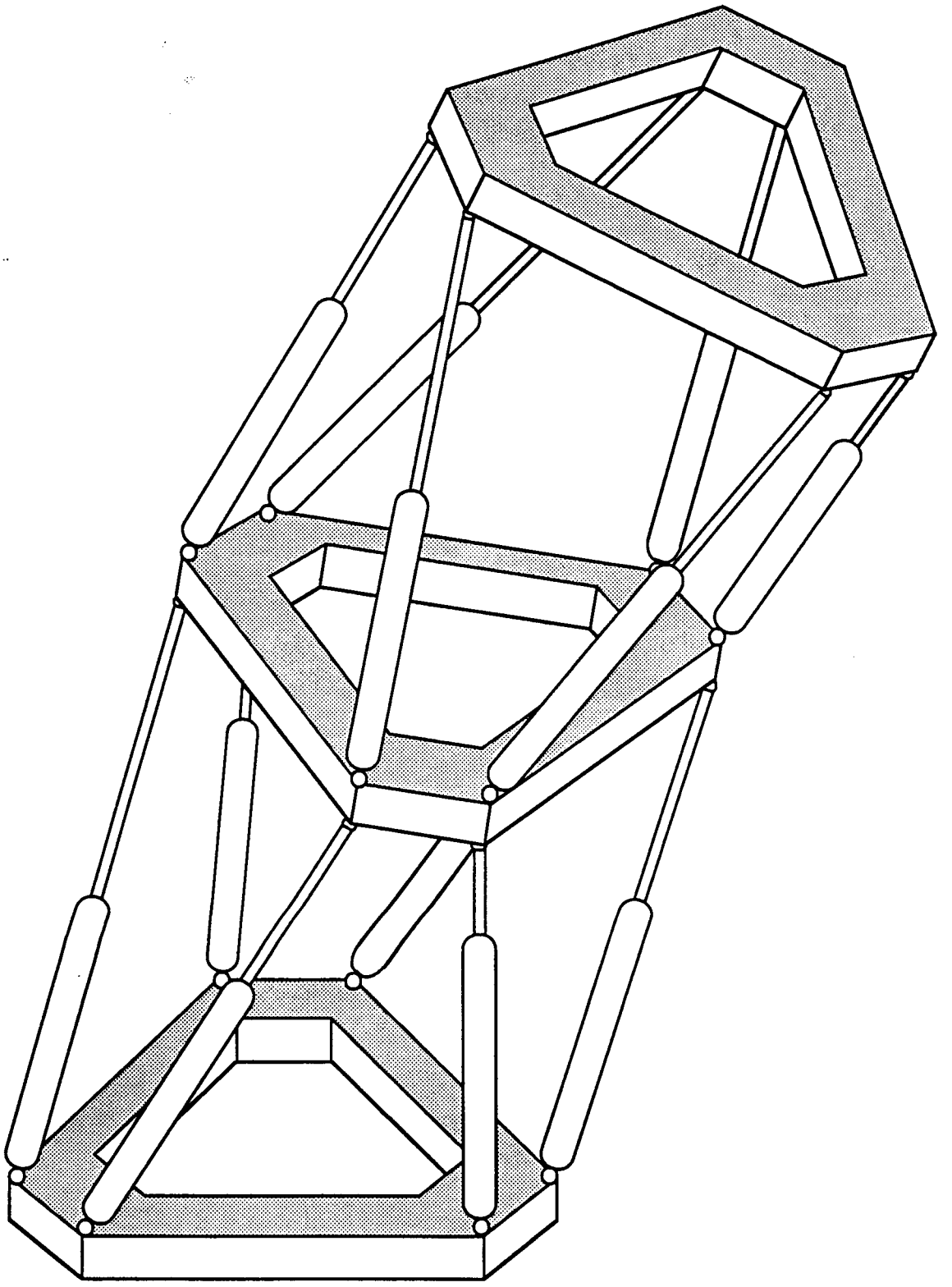


figure p3 - Two Segments

PJ-880610-013

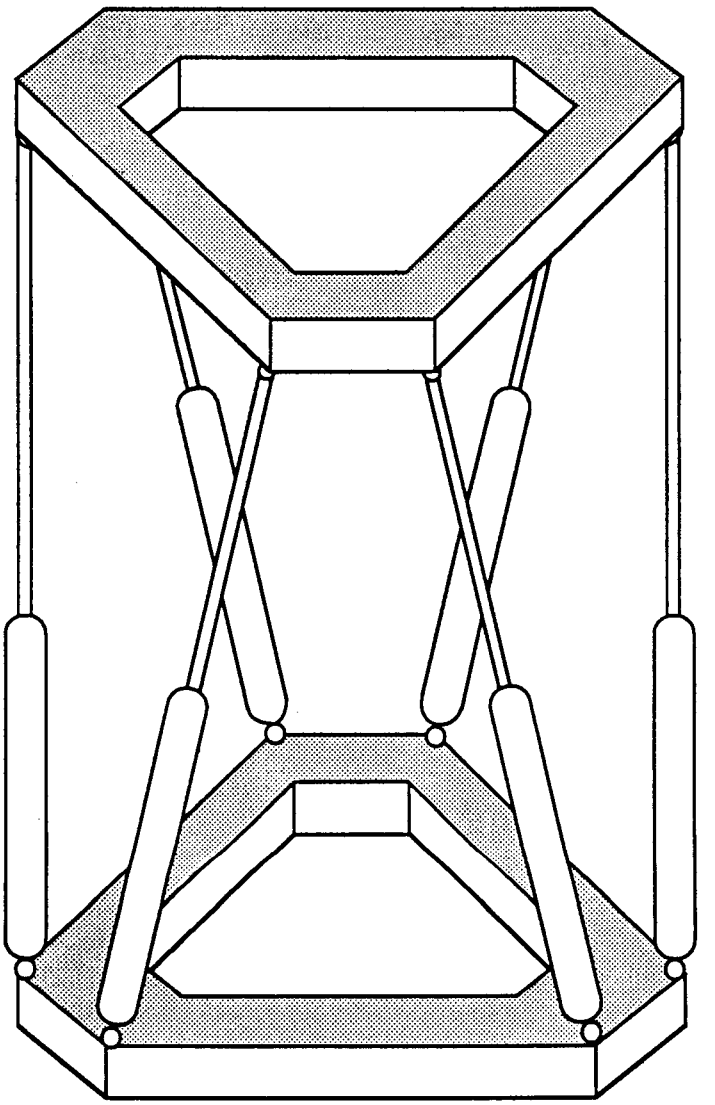
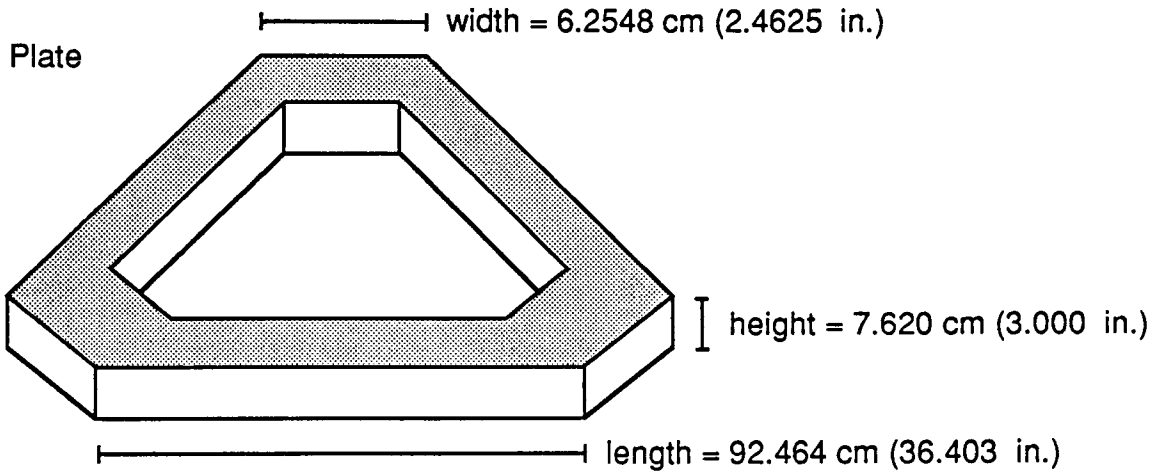
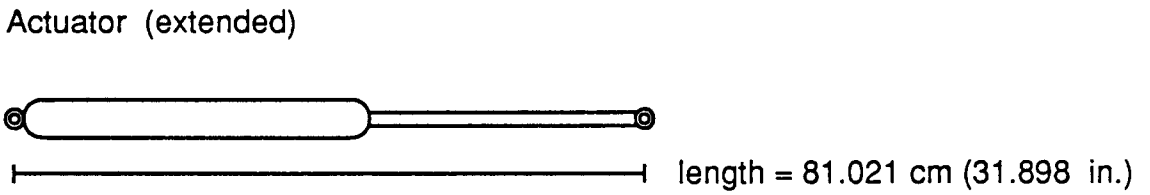


figure p4 - One Segment





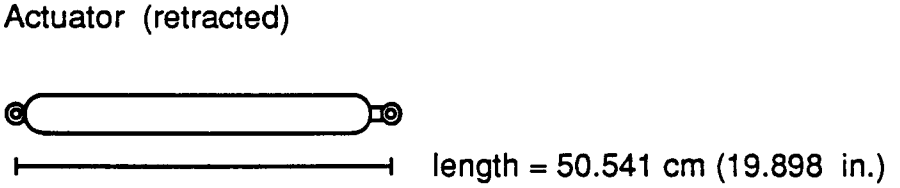


figure p5 - Parts of a segment

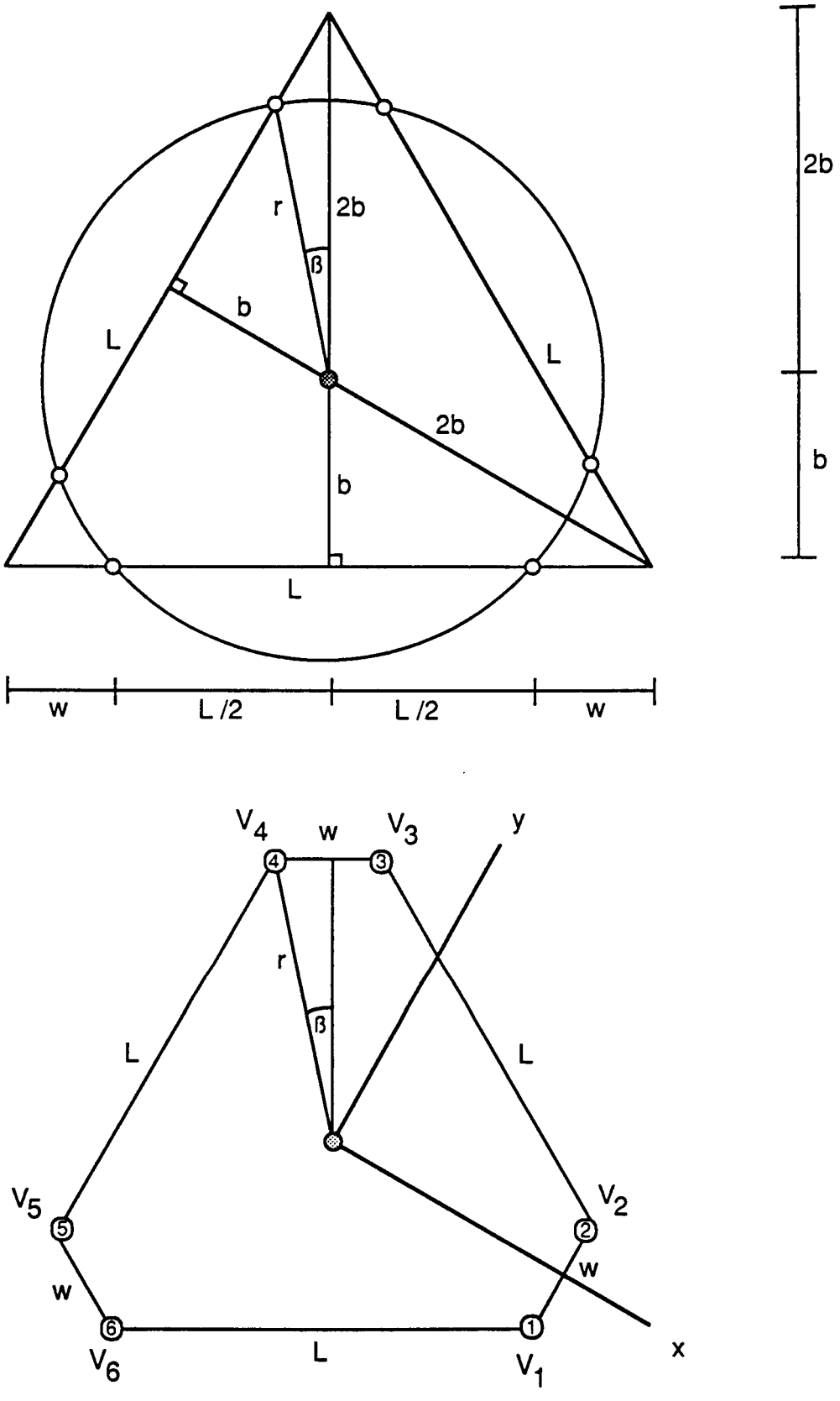


figure g1 - Triangular shaped Hexagon

PJ-880610-014

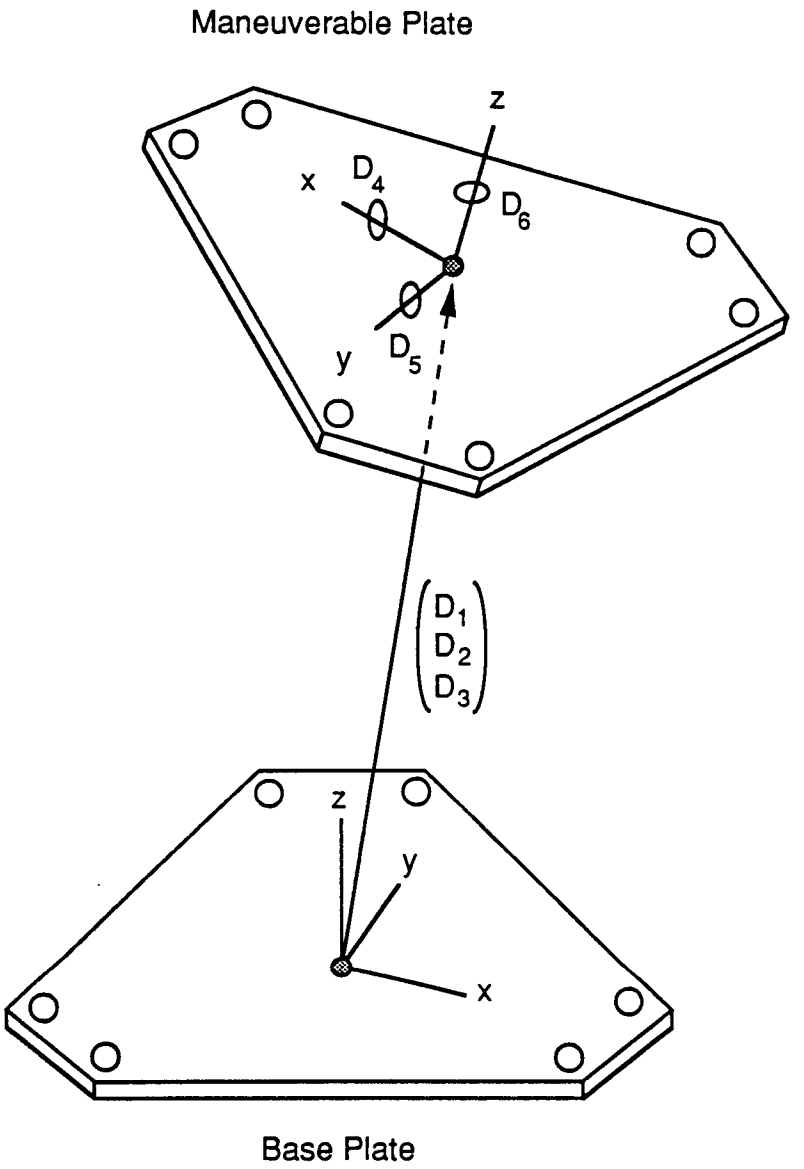
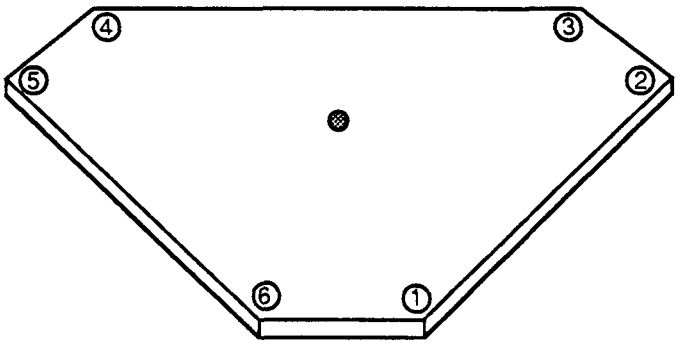
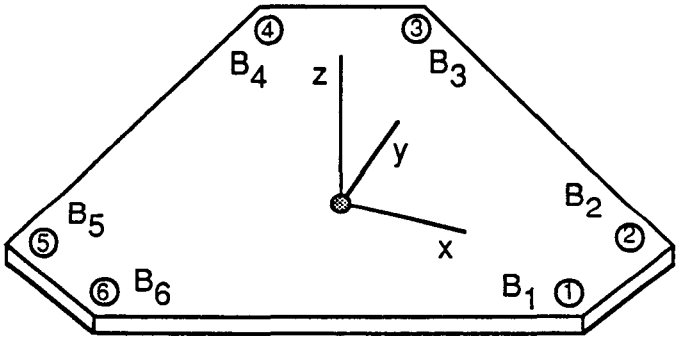


figure g2 - DOF-variables D, segment u

Maneuverable Plate





Base Plate

figure g3 - Coordinates Vertices Base Plate

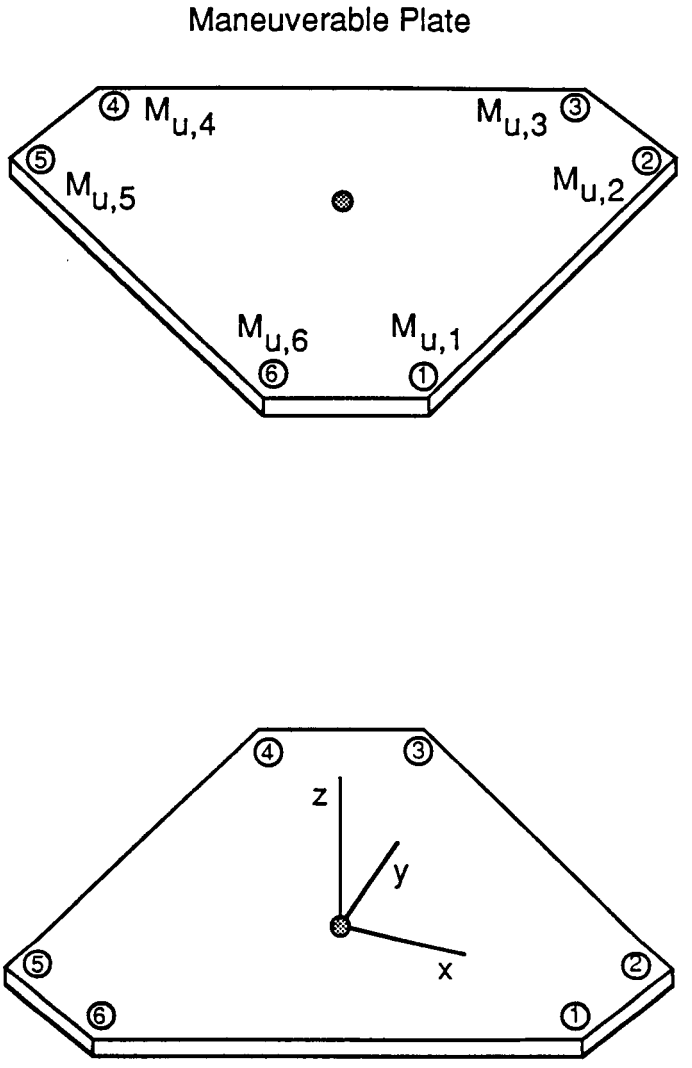


figure g4 - Coordinates Maneuverable Plate Segment u

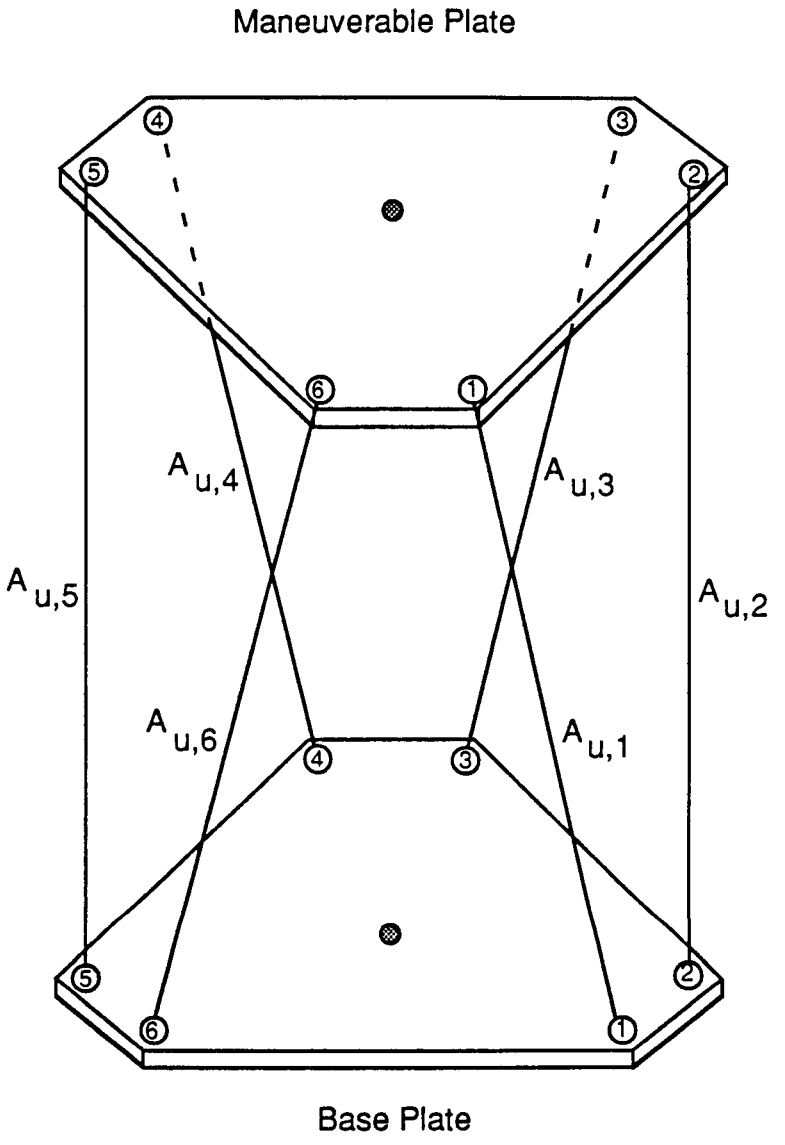


figure g5 - Actuator lengths A , segment u

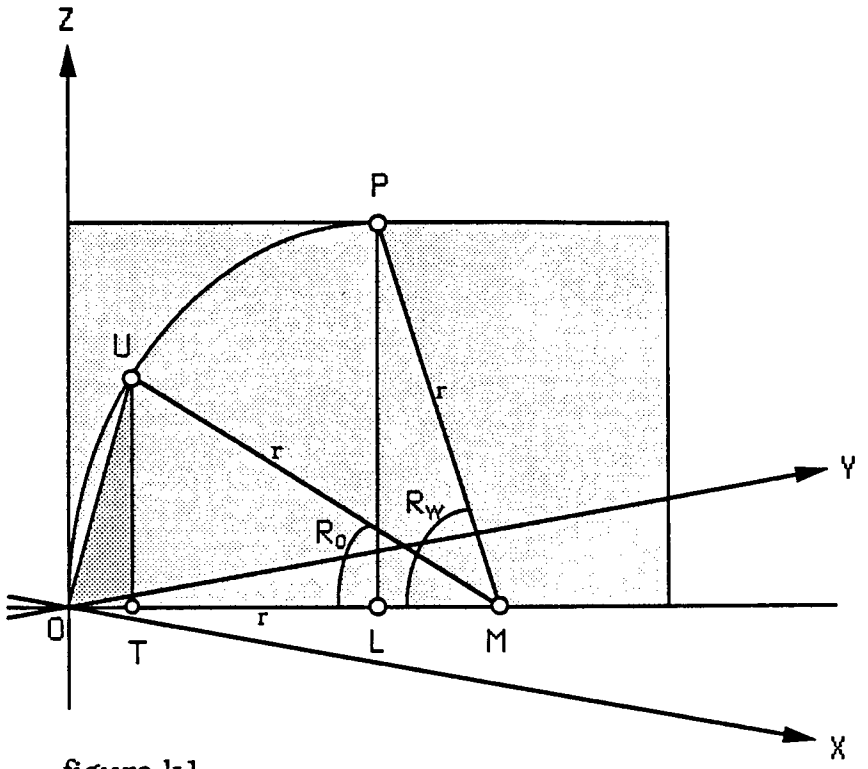


figure k1

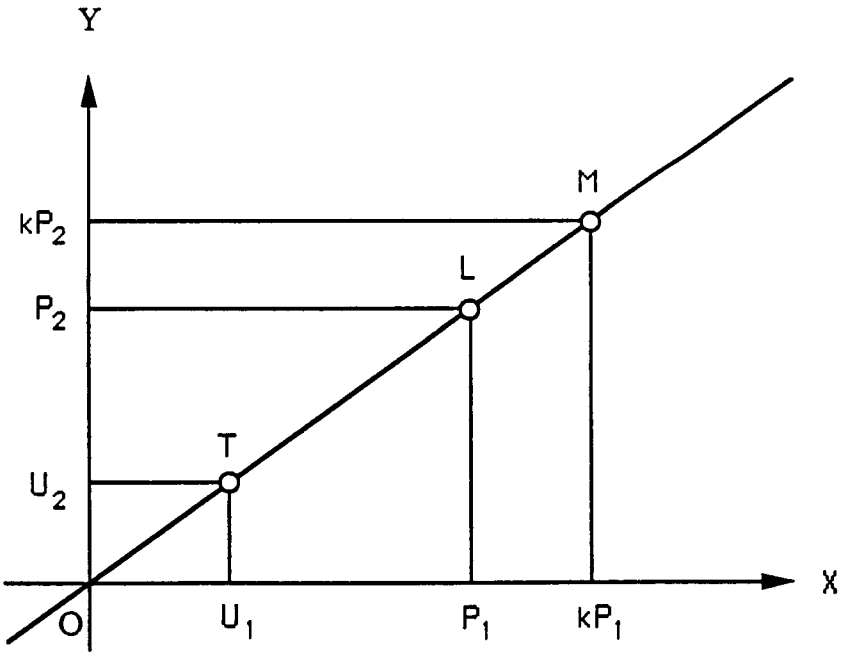


figure k2

PJ-880610-001

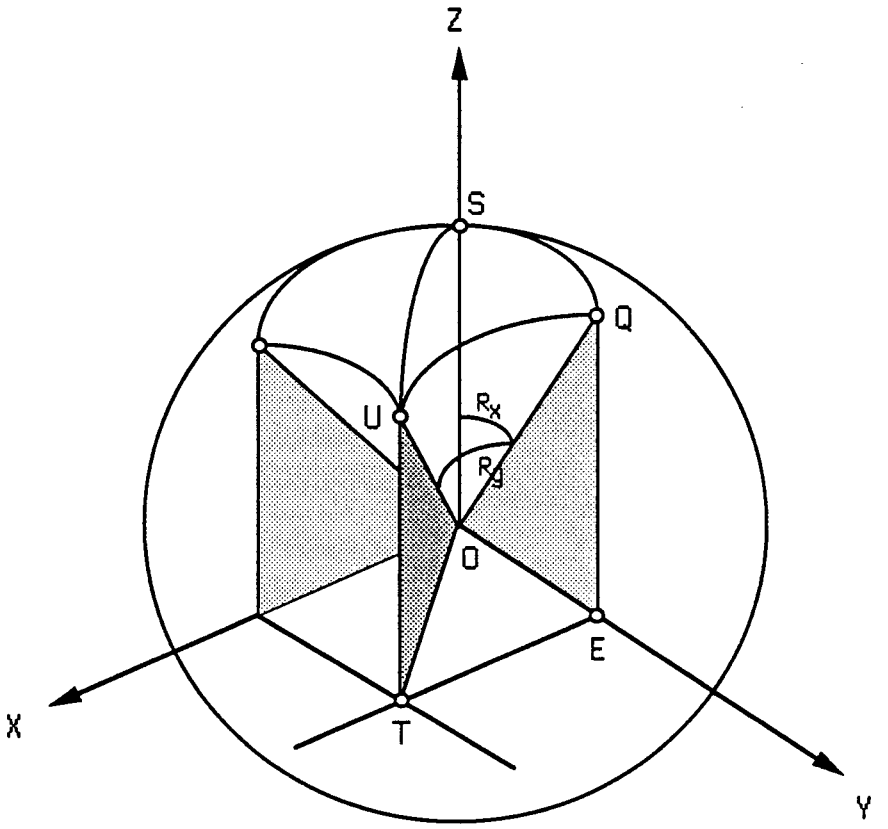


figure k3

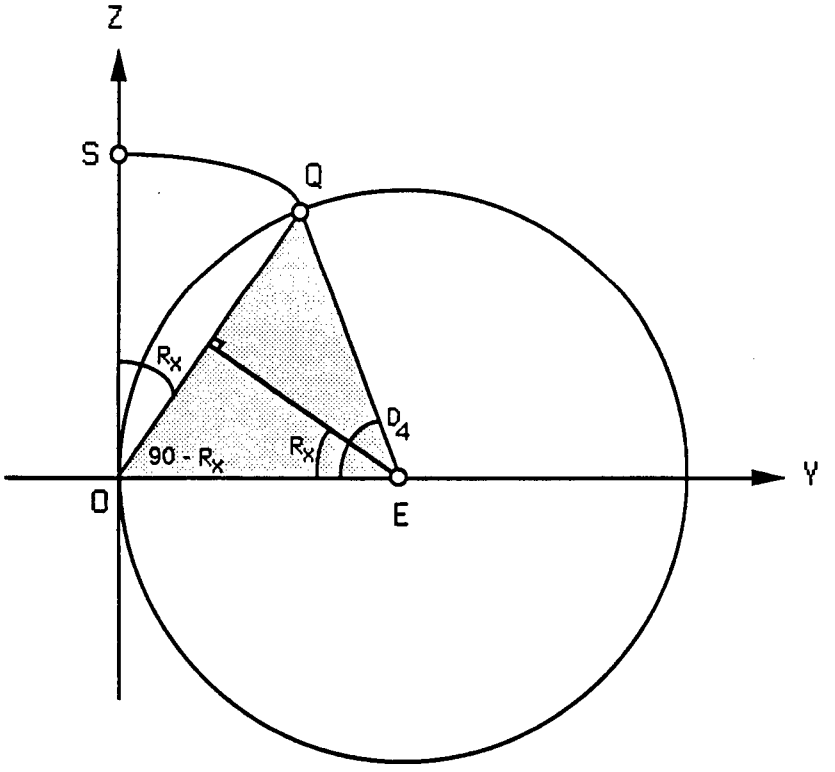


figure k4

PJ-880610-002

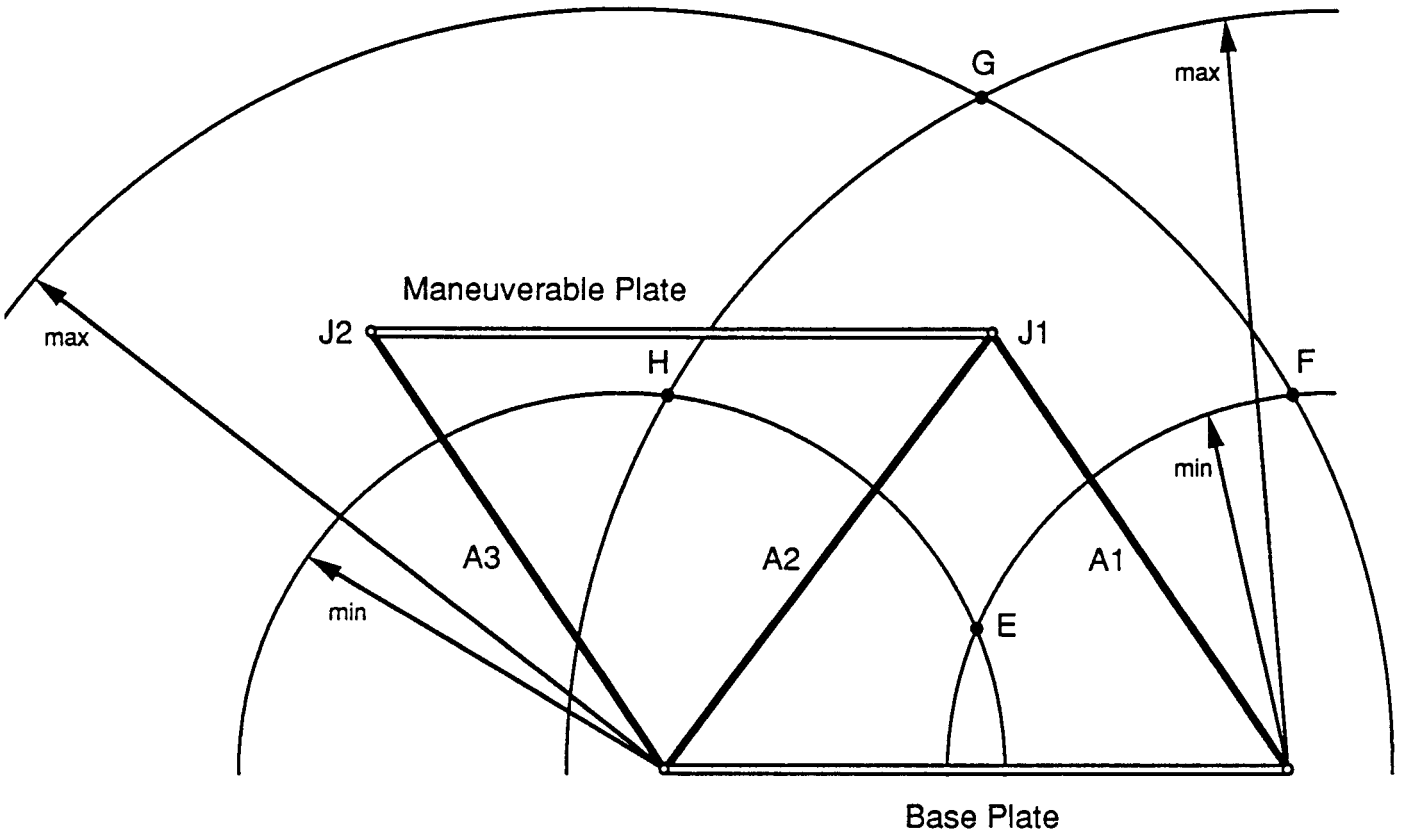


figure m1 - Segment in balanced position

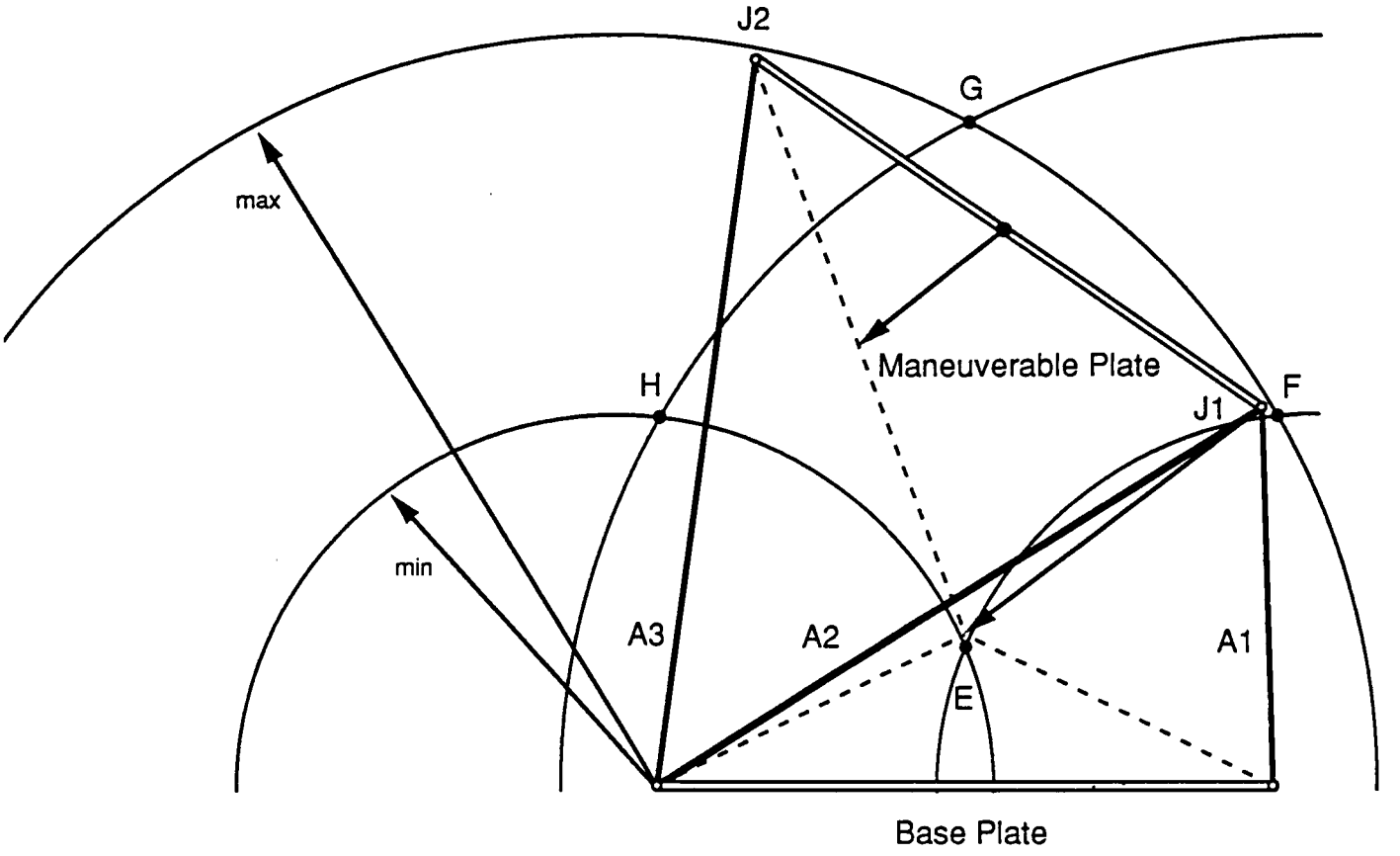


figure m2 - Maneuvering based on DOF variables

PJ-880610-009

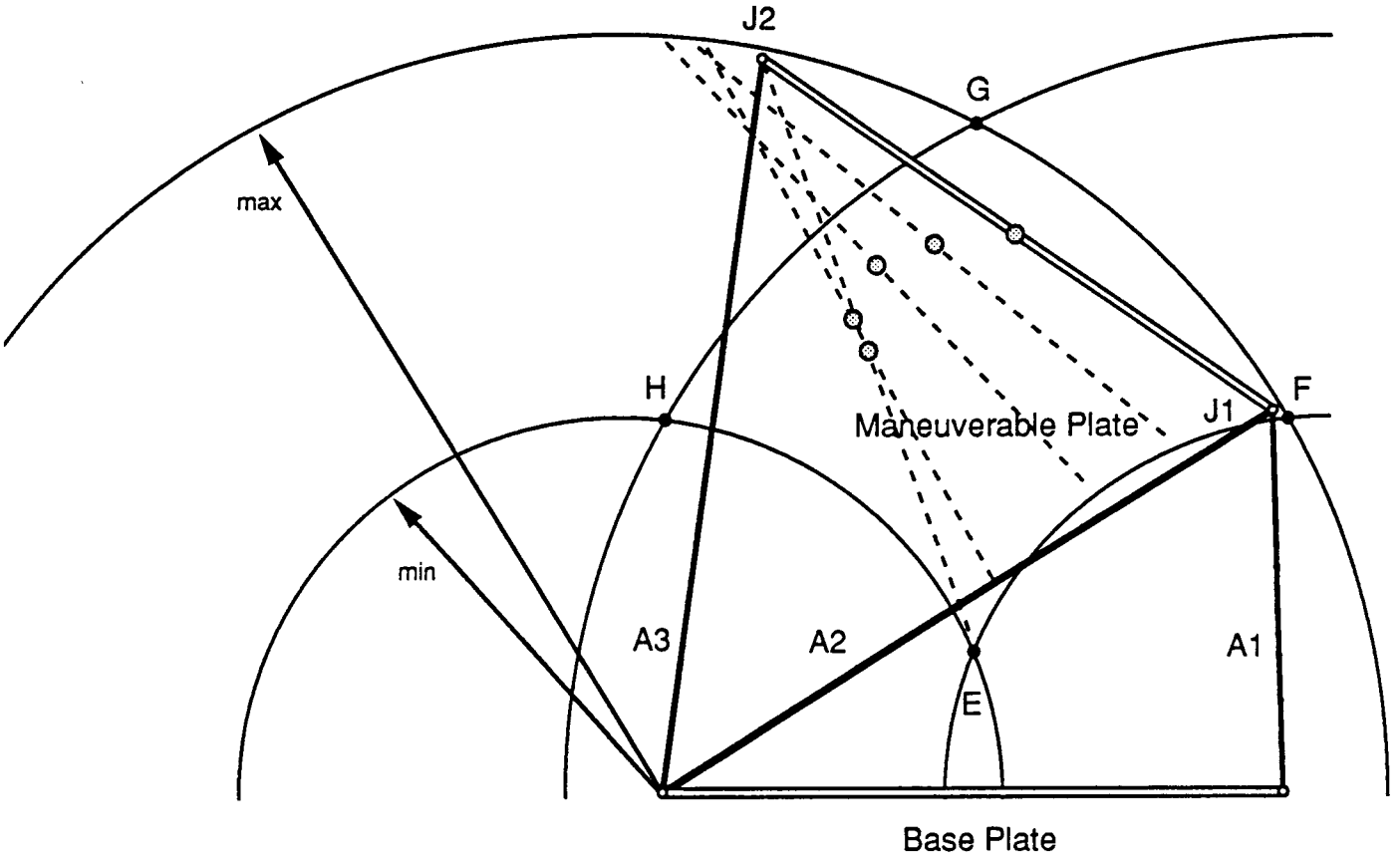


figure m3 - Maneuvering based on actuator lengths

PJ-880610-010