5-31-1990

Design of spatial linkages for dwell function generation

Donald F. Kelly
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

Title of Thesis: Design of Spatial Linkages for Dwell Function Generation

Donald F. Kelly, Master of Science in Mechanical Engineering

Thesis directed by: Associate Professor Raj S. Sodhi

Certain single-loop spatial linkages containing spherical pairs may be used as dwell function generators. Analysis and design of these linkages, however, may be too complex for the practicing engineer.

In this study a method is presented where the set of link lengths representing a linkage are determined from a smaller set of design parameters. Graphical relationships are also found between the design parameters and functional requirements for dwell, such as output function amplitude and dwell range. These graphs may be used as design charts, and design becomes the calculation of link lengths from design parameters chosen to satisfy the functional requirements.

This method is applied to RRSC, RSRC, and RSCP dwell linkages. Design parameters and design charts for these linkages are developed and presented here.

DESIGN OF SPATIAL LINKAGES FOR DWELL FUNCTION GENERATION

by

Donald F. Kelly

A Thesis

Submitted to the Faculty of the Graduate Division of the

New Jersey Institute of Technology

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Mechanical Engineering

May, 1990

APPROVAL SHEET

Title of Thesis: Design of Spatial Linkages for Dwell Function Generation

Name of Candidate: Donald F. Kelly

Master of Science in Mechanical Engineering, 1990

Thesis and Abstract Approved: _____ 5/9/90

Dr. Raj S. Sodhi Date

Associate Professor

Department of Mechanical Engineering

_____ 5/8/90

Date

_____ 5/9/90

Date

VITA

Name: Donald F. Kelly

Permanent address:

Degree and date to be conferred: MSME, 1990

Date of Birth:

Place of Birth:

Secondary education: Manalapan High School, 1981

Collegiate institutions attended	Dates	Degree	Date of Degree
Brookdale Community College	1983 1985	AS	1985
New Jersey Institute of Technology	1986 1988	BSME	1988
New Jersey Institute of Technology	1988 1990	MSME	1990

Major: Mechanical Engineering

Position: Designer

Foster Wheeler Corporation

Clinton, New Jersey, 08809

To my Mother and Father

ACKNOWLEDGEMENT

The author wishes to express his sincere gratitude to his advisor, Dr. Raj S. Sodhi, for his guidance and moral suport throughout this research.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

Page

CHAPTER 1

INTRODUCTION

1.1 Introductory Remarks and Definitions

A mechanism may be defined as "a mechanical device that...transfers motion and/or force from a source to an output." Linkages, or kinematic chains, are a subset of this group, and consist of rigid bodies (the links) connected by joints permitting certain relative motions to achieve the desired overall motion [16]. In the context of this thesis, the term linkage is limited to exclude such machine elements as gears, belts and pulleys, and cams; in other words, higher pairs are not used in linkages as they are defined here. Some typical examples of linkages would be the crank-slider mechanism and the planar four-bar mechanism.

Both of these examples are members of the class of planar mechanisms, the constituent parts of which are constrained to move in parallel planes. These devices are more easily analyzed using two dimensional mechanics, and so are more familiar and more commonly used than the more general class of spatial mechanisms. Spatial mechanisms are not constrained to planar motions, and thus they are more versatile than the planar mechanisms. The design of certain spatial linkages is addressed in this thesis.

One further remark, in relation to the above examples, may be made. These examples are types, or classes of linkages; the number of links and general configuration of the system are specified, but the particular dimensions of the component links are not. Choosing a general linkage class is known as type or number

synthesis, and the subsequent choice of the link dimensions is called dimensional synthesis. In this thesis, a linkage type or class is called a linkage configuration, while the result of a dimensional synthesis is called a particular linkage, or simply a linkage. All linkages are particular cases of some corresponding linkage configuration.

In many mechanism design problems, the output motion is required to be stationary for some portion of the input motion. This type of output is known as a dwell, and several machine elements, such as cams or geneva mechanisms, may be used to produce dwell motions. In some situations, and particularly at high speeds, these machine elements suffer from shock and dynamic effects. Linkages consisting of rigid bodies in permanent contact may in some cases offer a design solution that avoids these problems. A further advantage with linkages is that they are easier to manufacture compared to elements such as cams, and so are less expensive.

The drawback to using linkages for dwell applications is that a true dwell, where the output is absolutely stationary for a finite portion of the input cycle, is impossible for a linkage to achieve, since the output motion of a linkage is continuous in all derivatives with respect to the input motion. True dwell is not always necessary, however, and an approximate dwell (called a hesitation) can be produced by making all the derivatives of the output, up to a certain level, instantaneously equal to zero at the hesitation point [6]. This produces a motion that is instantaneously stationary at the point of hesitation, and nearly stationary nearby. In this thesis, all dwells are approximate, and the terms dwell and hesitation are used interchangeably.

1.2 Background

Although spatial linkages are more difficult to work with than planar linkages, over the past fifty years spatial linkage kinematics have been modeled successfully with a variety of techniques. K.H. Hunt has used analytic geometry [7], particularly curves of intersection between regions generated by the endpoints of open chains[8]; J. Rastegar has developed a similar method [15]. A. Wilhelm has applied projective geometry to spherical and other spatial mechanisms [27]. Other treatments include dual numbers and screw algebra [4], dual number quaternion algebra [28], matrices[3,16,22], and tensor methods [11]. The matrix technique is the basis of many commercial mechanism analysis programs because the technique is general and the lends itself well to numerical solutions[9,16,21]. Other computer methods, based on optimization techniques[25] and least-square methods[1], have application to synthesis of spatial linkages.

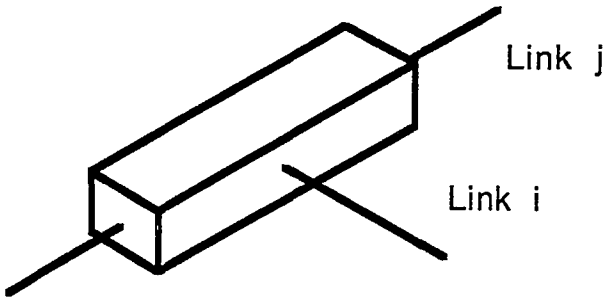
B.L. Harding studied the linkage motion requirements for producing a hesitation, or approximate dwell, and developed a technique for synthesis of dwell linkages using point-path mechanisms and path curvature theory [6]. Since then, Harding's techniques have been extensively developed for the planar case, and planar dwell linkage design has been automated by Kota, Erdman and Riley, using computer-based expert systems [12-14]. For the spatial case, K.H Hunt's geometric techniques have been applied by Hunt and Shrivastava to the problems of identifying and analyzing dwell linkages [18,19].

1.3 Nomenclature and Notation Explanation

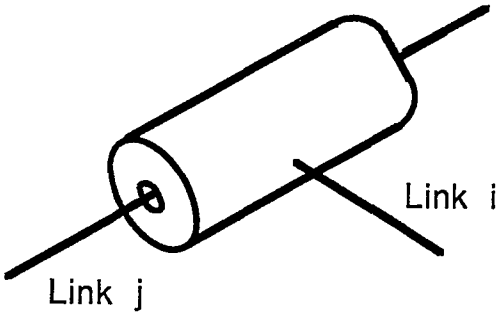
For the linkages studied in this paper, the lower pairs used are the revolute, or pin joint, the prismatic, or slider joint, the spherical or ball-and-socket joint, and the cylindrical joint; these pairs are represented by the capital letters R, P, S, and C, respectively. The links themselves are numbered, starting with 1 for the frame and continuing with the link connected to the frame by the pair containing the input variable.

Linkage configurations are named by the pairs that they contain, in the same order in which they connect the links, and a particular joint in the linkage is specified by subscripts indicating which links it connects. Thus the RRSC linkage contains a pin joint connecting the frame to link 2, another pin joint connecting link 2 to link 3, a spherical joint connecting links 3 and 4, and a cylindrical joint connecting link 4 to the frame. The second joint in this linkage is specified uniquely by R_{23}. In illustrations of linkages, joints are represented schematically as shown in Figure 1.

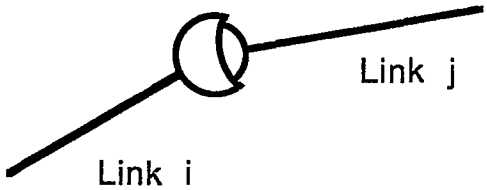
The primary mathematical entities used in this thesis are scalar variables representing link lengths and joint variables, vectors, and matrices. Link lengths are represented by lowercase italics, joint and other angles by Greek letters, vectors by boldface letters, and matrices are represented by capitalized Roman letters. When a link length is normalized, it will be represented by the roman capital equivalent of its original symbol; whether a capital letter represents a normalized link length or a matrix will be clear by the context in which it appears. Other scalar variables that may be introduced will use lowercase Roman and Greek letters.



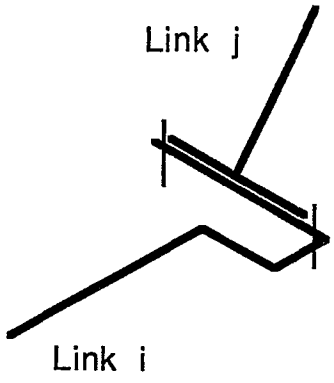
Prismatic Joint: P_{ij}



Cylindrical Joint: C_{ij}



Spherical Joint: S_{ij}



Revolute Joint: R_{ij}

Figure 1: Schematic Representation of Spatial Joints

1.4 Scope and Purpose of Thesis

The purpose of this thesis is threefold: first, to examine the kinematic necessities of design for dwell in spatial linkages; second, to adapt a method of kinematic analysis for use on a personal computer; and third, to study several spatial linkages with dwell potential, and from this develop a series of design charts to obtain suitable dwell motions from these linkages.

The scope of the first objective will be limited to position analysis only, and will attempt to answer the question: what are the necessary objectives when designing a spatial mechanism to produce dwell motion, beyond those needed for general mechanism design? Such design issues as transmission angles, branch problems, and the ability of the input crank to fully rotate are important, but are not uniquely germane to the dwell design problem. Separate criteria for identifying acceptable dwell designs, to be used in conjunction with the other elements of linkage design, will be presented here.

To perform the positional analyses implied above, a general method of modeling and analyzing the linkages studied must be available. The method of modeling linkages as transformations in homogeneous coordinates was chosen, and a library of subroutines was written for use in the general position analysis of any single-loop linkage. For many linkages, however, this matrix technique may be too elaborate, and a simpler alternative method, based on projective geometry, was used where possible.

The ultimate goal is to use the two proceeding objectives to produce something useful for the design of dwell linkages. Several spatial linkage configurations identified as having dwell potential were studied in terms of dwell criteria, and the results are

given in graphical form in this thesis. The mechanisms studied were the RRSC, the RSRC, and the RSCP linkages.

1.5 Discussion of Spatial Dwell Linkages

When linkages are classified according to function, they generally fall into three groups: point-path mechanisms, function generators, and motion generators. In the point-path mechanism, the objective is to have some point on a floating link follow a specified path, while for the motion generator, the entire rigid body motion of the floating link (or the moving lamina associated with it) is specified. The function generator produces a motion such that one joint variable for a joint connected to the frame is a specified function of a joint variable for another joint (the input) connected to the frame. The dwell linkages considered here are all function generators.

One property of the functions generated by linkages is that they are continuous in all their derivatives. Because of this property, dwells are only approximate, but the shocks associated with discontinuities in the function's derivatives do not occur. Further, under the conditions where the assumption of rigid links holds (ie. for sufficiently stiff links, or low inertial and other forces), these linkages avoid the dynamic effects, such as phase lag and spring surging, seen in cam and follower systems.

The main drawback to using linkages for dwell, aside from the approximate nature of the dwell motion, is that since the derivatives (especially jerk) of the output are low near the dwell point, large accelerations may occur in the output function as the motion resumes.

The method of identifying spatial dwell linkages may be more easily broached by considering methods used in the planar case [6]. To produce a planar dwell linkage, a point-path mechanism is synthesized so that the path contains a portion that is nearly straight or that has a nearly constant curvature. (Cusps and inflection points in the path may also be used to produce dwells, but the above cases are sufficient for illustration purposes.) A second loop is constructed as shown in Figure 2, so that a link that has the same length as the radius of the curve is connected by a pin to the moving pivot of the new crank, or a slider joint is tangent to a nearly straight-line portion of the curve. The new crank will exhibit dwell motion when the point is in the linear or constant-curvature portion of its path. This is equivalent to finding a second (open-loop) point path linkage, whose path is constrained by the path of the closed loop point to experience a dwell condition.

For spatial linkages, an analogous method was developed by Hunt to investigate the motion of possible configurations. In this method[8], a point on the end of an open loop linkage (or kinematic chain) generates a curve, surface, or volume of all the points it can reach. The intersection of this generated region with that from a point on another open loop creates, under favorable conditions, a curve of intersection; if the two open loops are joined at a points of intersection by a spherical joint, a closed loop is formed, and the locus of possible positions of the spherical joint is the curve of intersection. Geometric analysis of this curve can tell much about the mechanism thus constructed.

In terms of dwell mechanisms, the design strategy is to find the conditions under which, when the output joint variable is held motionless at the dwell point, the generated surface of the output chain contains a path that osculates (intersects another curve at three or more infinitesimally separated points, or in other words has a common

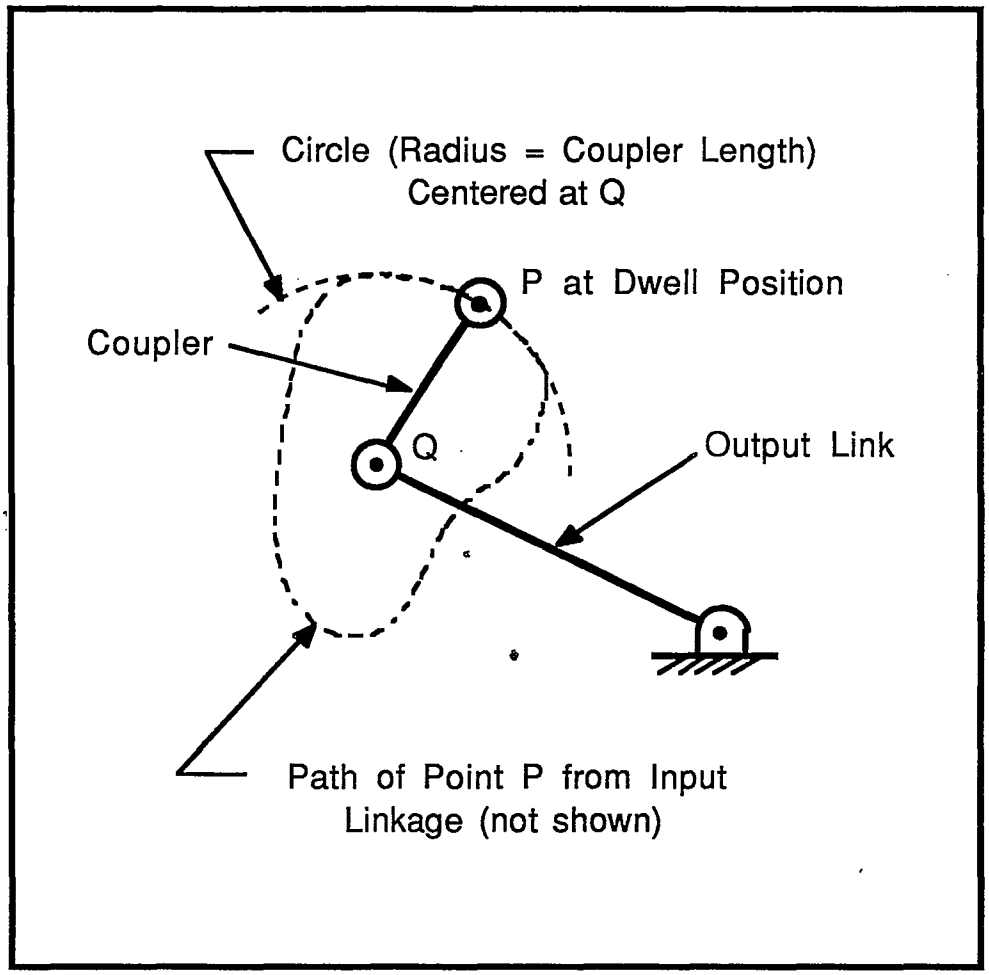


Figure 2: Method of Developing Planar Dwell Linkages

tangent and center of curvature at a point of intersection with some other curve) with the curve of intersection (the path of the spherical joint). Thus a dwell linkage synthesized in this fashion will have, in general, at least one spherical joint, and will, in addition to constraints on its dimensions to assure full crank range of motion, transmission effectiveness, and so on, have additional constraints on its dimensions to assure the presence of this osculation[19]. Again, the linkage is synthesized by making an open-loop path generator conform to a path that produces a dwell, but in the three dimensional case the intersection of the two open chains is what produces the path, rather than an input point-path mechanism, and so only one loop is necessary.

In designing for approximate dwell motion, there are three new functional requirements. The first of these is the overall shape of the output function, which would include the number and placement of the hesitations. The second functional requirement is the total span of motion traversed by the function, and can be thought of as the amplitude of the output function. The third, which must be specified for each dwell, is the range, in terms of input motion, that the output remains approximately stationary. Since the dwell is approximate, a tolerance is applied: the accuracy of the dwell is the percentage of span that the output deviates from its position at the exact dwell point; and dwell ranges are given meaning by the application of a dwell accuracy. Thus one may meaningfully speak of a "forty degree dwell range, with an accuracy of one percent," which would mean that the input crank rotates 40 degrees while the output variable deviates less than one percent of the total span from the dwell position.

For a dwell linkage configuration, therefore, a study must be made of the overall output function shape, the total span of the output, and the dwell ranges for given

accuracies, for all possible sets of link dimensions producing dwell conditions. This would give the designer the information he needs to choose intelligently from among the possible configurations, and develop the linkage he needs for his application.

1.6 Axiomatic Design

Concepts such as functional requirements and design parameters are part of a new theory of design, known as axiomatic design and developed by N. P. Suh [23, 24]. In this theory, functional requirements are a set of independent requirements that the completed design must meet, and design parameters are a set of variables relating to the design choices available in any particular design; the vector of functional requirements is a vector function of the design parameter vector.

The theory of axiomatic design is based on two axioms postulated by Suh. The first, called the Independence Axiom, states that designs in which the design parameters are independent of each other, with respect to the vector function relating them to the functional requirements, are superior to designs in which the design parameters are not independent. The Independence Axiom has a direct bearing on the approach taken in this thesis, as this approach deals with the relationships between functional requirements and design parameters. The second axiom, which is known as the Information Axiom and is less important to this thesis, states that of the designs satisfying the Independence Axiom, the design requiring the least descriptive information is the best design.

Some consequences of the Independence Axiom may be stated in the following manner: for functional requirement vector **F** and design parameter vector **P**, related by

vector function v such that $\mathbf{F} = \mathbf{v}(\mathbf{P})$, then the best designs are those in which the Jacobian of **v** is a diagonal matrix, since each design parameter affects exactly one functional requirement and the parameters are thus independent. Designs meeting this requirement are known as uncoupled designs. Less desirable are decoupled, or quasi-coupled, designs, in which the Jacobian of **v** is a triangular matrix; in these designs the parameters are not independent, but by solving for the parameters using forward or back substitution, some independence may be maintained. The least desirable design is the coupled design, where the design parameters cannot be made independent.

A consideration of these results and the Information Axiom leads to the conclusion that the best designs have the same number of functional requirements as design parameters.

The Axiomatic Design approach supplies a conceptual framework for the classification and evaluation of dwell linkages. The functional requirements for dwell motion are given above; a method of analysis will be presented below to develop design parameters for dwell linkages.

CHAPTER TWO

ANALYSIS OF THE SPATIAL MECHANISMS

The analysis of a linkage configuration with dwell potential must start by finding the constraints necessary to assure a properly functioning linkage, and those necessary to assure a dwell condition; these constraints are used to develop design parameters for the dwell linkage configuration. Then for the particular linkages allowed by these constraints and prescribed by the design parameters, a position analysis must be made to find the actual values for the functional requirements. The first portion of the analysis is performed by using Hunt and Shrivastava's analytical geometry techniques[8, 18, 19]. The second portion is performed either through the matrix-numerical position analysis or the projective geometry technique mentioned above.

For a given linkage configuration, the information, such as link lengths and offsets, may be represented as a vector in a "linkage-" or "design space," with one component for each link dimension or offset. Particular linkages may then be represented as points in this design space. The constraints assuring proper linkage function are usually inequality constraints, so they define regions in the design space in which are found acceptable linkages. The constraints assuring dwell motion are usually equality constraints, so they reduce the number of independent variables needed to find the acceptable dwell linkages. The sets of acceptable dwell linkage vectors are curves or surfaces in the design space, and are vector functions of some new set of parameters, which themselves usually have some physical meaning in terms of the

linkage configuration. The parameters may also be considered components in a new, and smaller, "parameter space," the points of which map, through the vector function, into the dwell linkage points in the design space. The identification of the constraints and the development of the parameters and vector functions depend on the particular linkage configuration considered, and are left to the sections dealing with each configuration.

Once the parameters, mapping function, and constraints are known, dimension synthesis becomes simply choosing a point in parameter space, and checking that the resulting point in design space is in the acceptable region. Design for dwell is thus the task of choosing a linkage configuration and finding a point in its parameter space to meet the functional requirements. In general, the first functional requirement, the overall shape of the output function, is a function of the linkage configuration, while the span and the dwell ranges are functions of the parameters.

Once the configuration is chosen to meet the first functional requirement, the designer may consult some mapping of parameters to the span and dwell ranges to find the linkage fitting his requirements. In fact, the designer need not even consider the span of the output at this point since the output function may be cascaded through an amplification mechanism (pulley, rack and pinion, or another linkage) to scale the original output. Thus the primary concern of the designer is the dwell range.

The dwell range as a function of the parameters must at this point be found from a positional analysis of the linkage as the input crank is rotated. One analysis method that may be used is based on the well known matrix method, as used by such commercial packages as IMP and IDEAS, and presented in [16]. A brief recapitulation is presented here.

A point (x, y, z) is represented in homogeneous coordinates by the vector (column matrix) $[x, y, z, 1]^T$. The transformation of a point's coordinates from one coordinate system to another by translation or rotation may be performed by an appropriate premultiplication by a 4x4 transformation matrix. In other words, for a point P, if $\mathbf{P}_b$ is the homogeneous coordinate vector of **P** in coordinate system B, and $\mathbf{P}_a$ is the vector in system A, then $\mathbf{P}_a = S_{ab}\,\mathbf{P}_b$, where S_{ab} is the matrix used to move from system A to system B. A sequence of transformations can be represented by the product of the individual transformation matrices, and is called a concatenation of the transformations.

A joint connecting two links may be represented as one or more coordinate transformations, where the new coordinate system lies on the new link and the old system lies on the old link. A kinematic chain would then be represented as a concatenation of the joint transformations, and a closed loop would also be represented by a concatenation of the link transformations, but with the important property that the product of the transformations must be the 4x4 identity matrix (since the final link returns to the starting point). Some examples of useful joints are given here for illustration.

In Figure 3 a cylindrical pair is shown connecting Link i with Link j. The old coordinate system (X_0, Y_0, Z_0) lies along Link i with the origin coincident with the pair at the far end of the link, and the new coordinate system (X_2, Y_2, Z_2) lies on Link j, with the origin coincident with the Cij pair. The transformation for the cylindrical pair is a concatenation of two transformations. The first transformation is a translation L_i along the X_0 axis to the C_{ij} pair, then a rotation θ about the translated Z_0 axis to reach the interim (X_1, Y_1, Z_1) coordinate system. The second transformation is a translation T along the Z_1 axis to the origin of the new coordinate system. The joint variables for

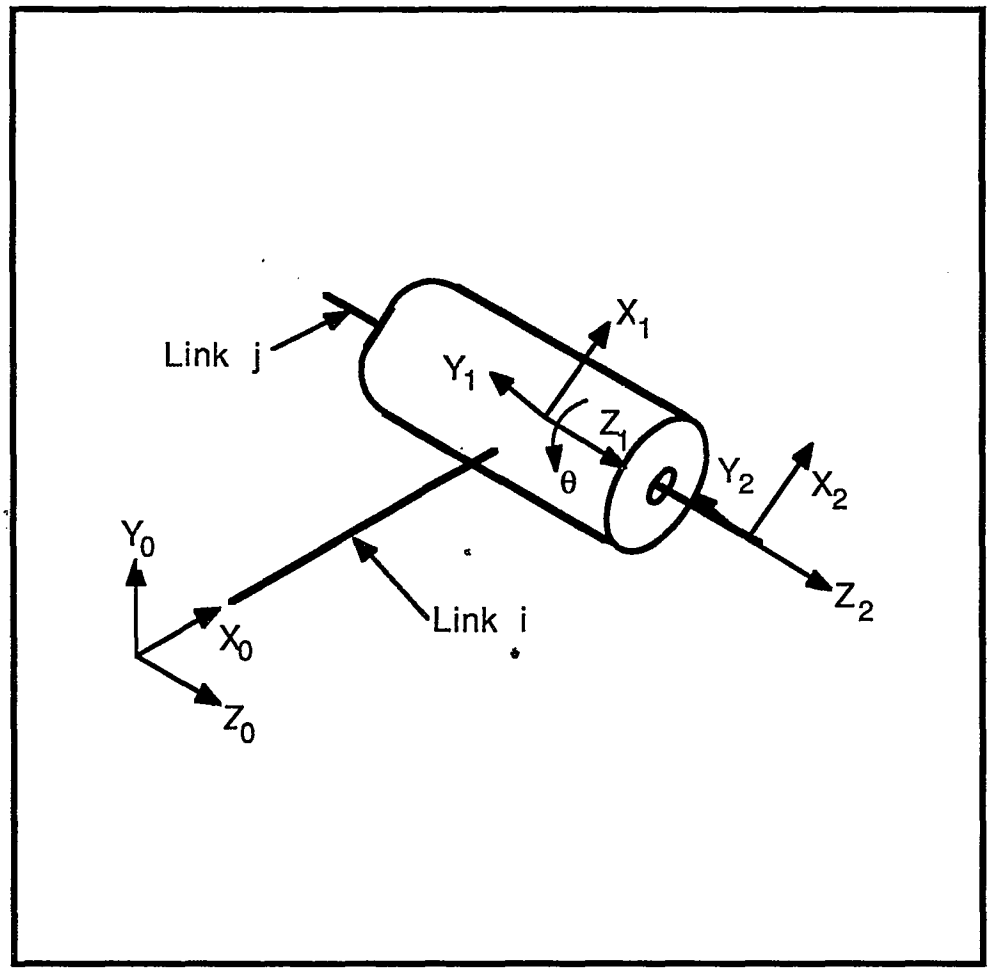


Figure 3: Modeling of Cylinder Joint with Two Transformations

the C_{ij} pair are θ and T. Revolute pairs may be modeled using only the first transformation, and prismatic pairs can be modeled using a θ of zero.

A spherical pair S_{ij} is shown connecting Links i and j in Figure 4. This joint is modeled by considering it to behave as three revolute pairs as shown in Figure 5. This joint is modeled with three transformations, corresponding to the three joint variables of the spherical pair or the three revolute pairs. The first transformation translates along Link i from the old origin (X_0, Y_0, Z_0) to the center of the sphere, then rotates θ_1 about the new Y axis. The second transformation is a rotation of θ_2 about the X axis, and the third transformation is a rotation of θ_3 about the Z axis. In figures 4 and 5 only the original and final coordinate systems are shown, and the rotations θ_1 and θ_3 are made equal to zero, to simplify the presentation. Other concatenations of three transformations may be used to represent this pair, depending on the configuration of the three revolute pairs used to represent it.

The position analysis of the single closed loop linkage becomes a problem of finding the values for the unspecified (dependent) joint variables for the given input values. This, in turn, is actually solving the problem S - I = 0, where S is the matrix product of all the joint transformation matrices, and is a nonlinear matrix function of all the joint variables, and I is the identity matrix. This is solved using the an adaptation of the iterative Newton-Raphson root-finding technique. This iterative technique depends on using partial derivatives of S with respect to each of the joint variables, and finds successively closer approximations to the true solution, given a sufficiently close initial guess.

These techniques have been implemented in a group of subroutines that together form a way of modeling and determining the position of any single loop linkage. These

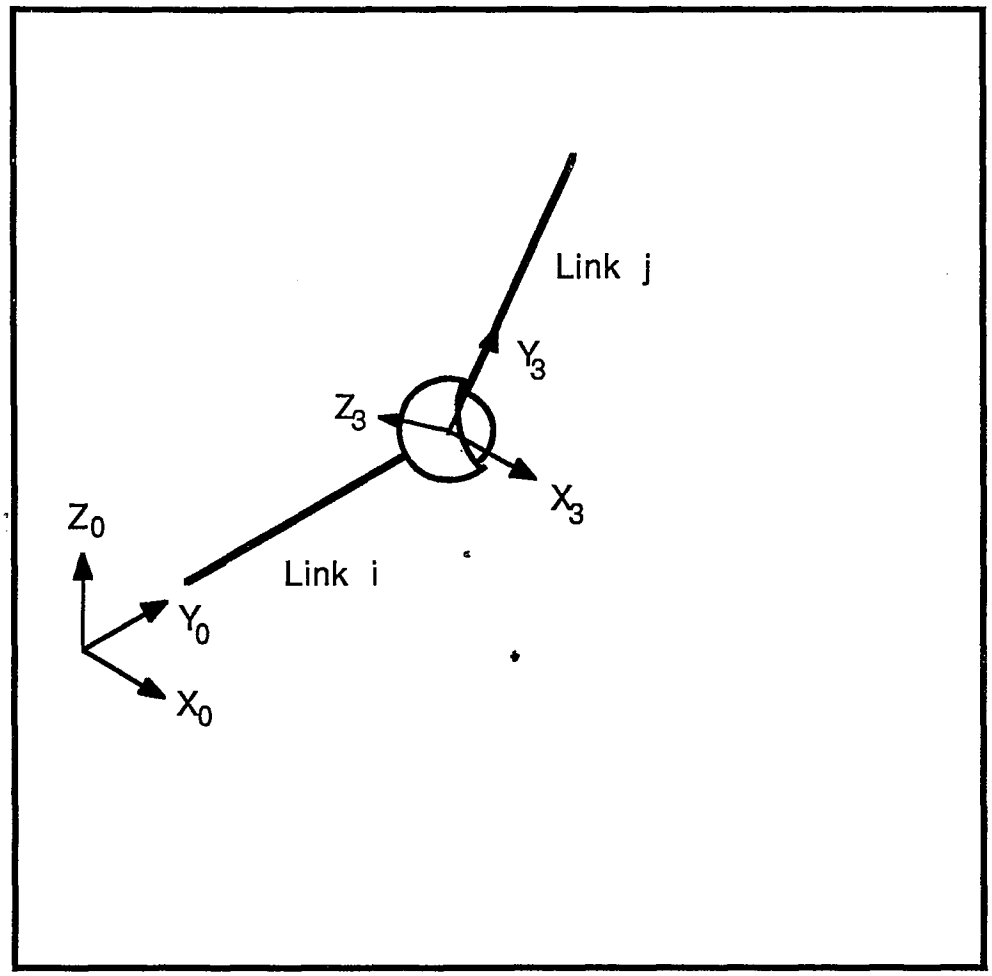


Figure 4: Initial and Final Coordinate Systems for Spherical Joint

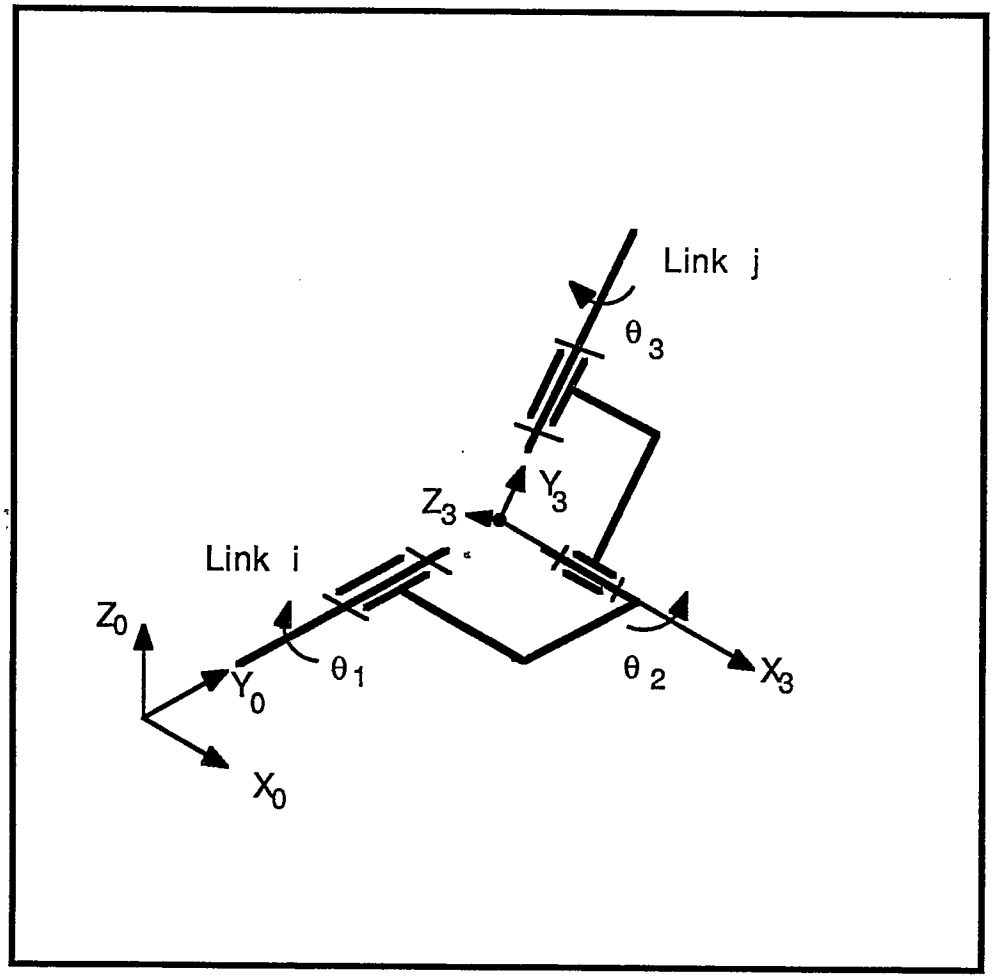


Figure 5: Transformation Modeling of Spherical Joint with Three Revolute Pairs

routines were written in THINK C on the Macintosh, and are based on the use of the matrix data types and routines in the commercial NuTools Numerical library for THINK C, as well as a series of "code primitives" that perform the elementary manipulations. A listing of these routines, along with a more complete description of their function and the techniques on which they are based, may be found in Appendix A.

A second method of position analysis, which is based on projections of the linkage onto convenient planes to reveal simple geometric relationships, may also be used on the spatial linkages presented here. This technique in effect reduces the spatial analysis to a set of planar analyses; its major advantages over the matrix technique are the simplicity of its application and the ease and speed of solution of the resulting models. The projective geometry approach does not lend itself easily to a universal modeling technique, however, so the technique must be re-derived for each linkage. Because it is different for each linkage, and because it is best demonstrated by example, the projective geometry approach will be presented when it is applied. The projective geometry approach was used wherever it was convenient, and the matrix technique was only used to demonstrate its application.

2.1 Tools Used

As was mentioned earlier, the matrix analysis technique was implemented on an Apple Macintosh Plus, and was coded in THINK C version 4.0 with the NuTools Numerical version 1.2 numerical routine library. The mapping function between design parameters and link lengths for a configuration is a nonlinear system of equations, and

the projective geometry technique also leads to systems of nonlinear equations; these were manipulated using TK Solver Plus version 1.1, an equation solving program, on the Macintosh Plus. This program can solve for unknowns in systems of equations, and can automatically apply a Newton-Raphson root finding algorithm if it cannot directly solve for an unknown variable. Most of the data for the design charts were generated by TK Solver models for the mechanisms. The design charts themselves were created by TempleGraph, a graphing program available on Sun workstations in the NJIT Computer-Aided Engineering Laboratory. Other drawings were created in MacDraft on the Macintosh.

CHAPTER THREE

APPLICATION TO SOME DWELL LINKAGES

The foregoing analysis was applied to three spatial linkages with dwell potential. Two of these, the RRSC linkage and the RSRC linkage, were studied in previous literature [18, 19], and the third, the RSCP dwell linkage, is introduced here.

3.1 The RRSC Mechanism

The RRSC mechanism is the most complex linkage configuration considered in this thesis. The overall configuration is as shown in Figure 6. Link 1 is the frame, the revolute joint R_{12} is the input crank, and the cylinder C_{41} is the output. There are two possible output variables, corresponding to the two joint variables of C_{41}. The major dimensions are given as crank length q, Link 3 length b, Link 4 length r, and the line of action of C_{41}, which is parallel to the Z-axis and offset from it by p vertically and h in the -X direction. All link lengths and offsets are considered to be greater than or equal to zero in this thesis. In this application, the link lengths p, r, b, and h are normalized with respect to crank length q to become P, R, B, and H, which are used as the coordinates of the RRSC linkage space. This linkage configuration was studied extensively by Hunt and Shrivastava [18, 19], who identified the conditions for proper linkage function and dwell motion.

The RRSC linkage may be split into two open kinematic chains by removing the S pair. The endpoint of the $R_{12}R_{23}$ chain produces a torus, with major diameter of Q=1

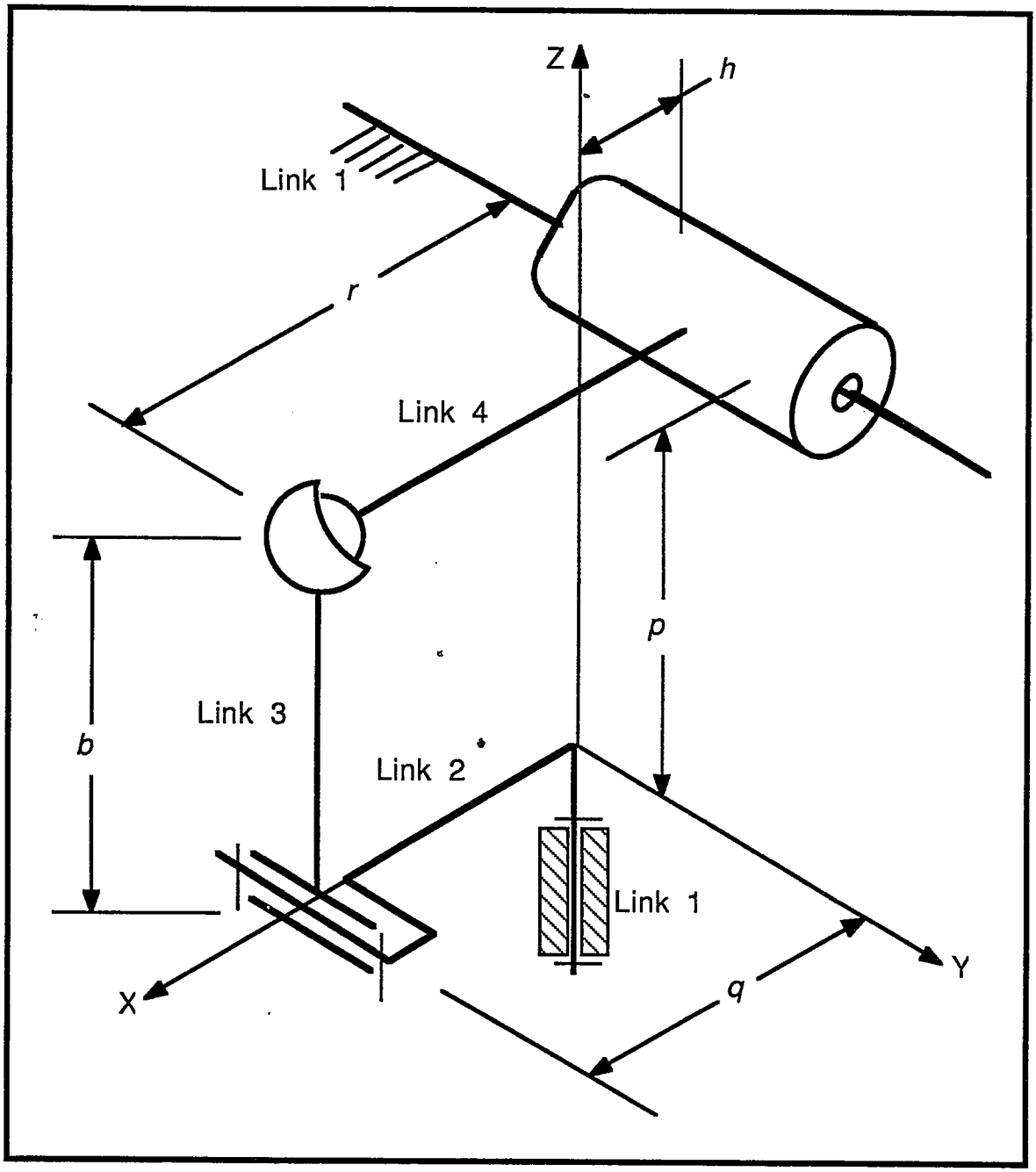


Figure 6: RRSC Configuration, with definitions of Link Lengths

and minor diameter B, as its reachable space. The cylindrical joint C_{14} produces an infinite cylinder of diameter R.

This general linkage configuration gives rise to four different dwell linkage configurations characterized by the number and position of dwell points: a single dwell point on the angular output variable, two identical dwell points 180° apart on the angular variable, two identical dwell points 180° apart on the translational output variable, and four dwell points 90° apart, alternating between translational and rotational dwells. These configurations are labeled here as RRSC.1ad, RRSC.2ad, RRSC.2td, RRSC.4d, respectively, for convenience. All dwell points in these linkages are at the the extrema of the output functions.

Hunt and Shrivastava have identified the necessary conditions for dwell in these linkages [19]. For the RRSC.1ad configuration, the equality constraint is given as:

$$R^2 = (1 + H)^2 + (P - B)^2 \qquad (1)$$

For RRSC.2td, the constraints are:

$$H = 0 \qquad (2)$$

$$BP/(R - P) = (P^2 + 1)^{0.5} \qquad (3)$$

The RRSC.2ad constraints are (1) and (2), and the RRSC.4d constraints are (1), (2), and (3). Thus the RRSC.4d configuration, which has four link dimensions and three

equality constraints, would be expected to have one design parameter; the RRSC.2ad and RRSC.2td would each have two design parameters, and the RRSC.1ad configuration would have three design parameters.

Full crank rotation is the only general constraint considered in this thesis. The conditions assuring crank rotability will be considered separately for each of the four dwell configurations.

The RRSC linkage may be studied using the projective geometry technique. By projecting the link lengths onto the XZ-plane (Figure 7), the following relationships may be found:

$$\cos\alpha\,(1 + B\sin\beta) = R\sin\gamma - H \qquad (4)$$

$$B\cos\beta + R\cos\gamma = P \qquad (5),$$

where α is the crank angle, β is the joint variable for R_{23}, and γ is the angular joint variable for C_{41}, as shown in the figure. These may be solved simultaneously for β and γ, and a projection onto the YZ-plane (Figure 8) produces an equation for the translational variable T for C_{41}:

$$\sin\alpha\,(1 + B\sin\beta) = T \qquad (6)$$

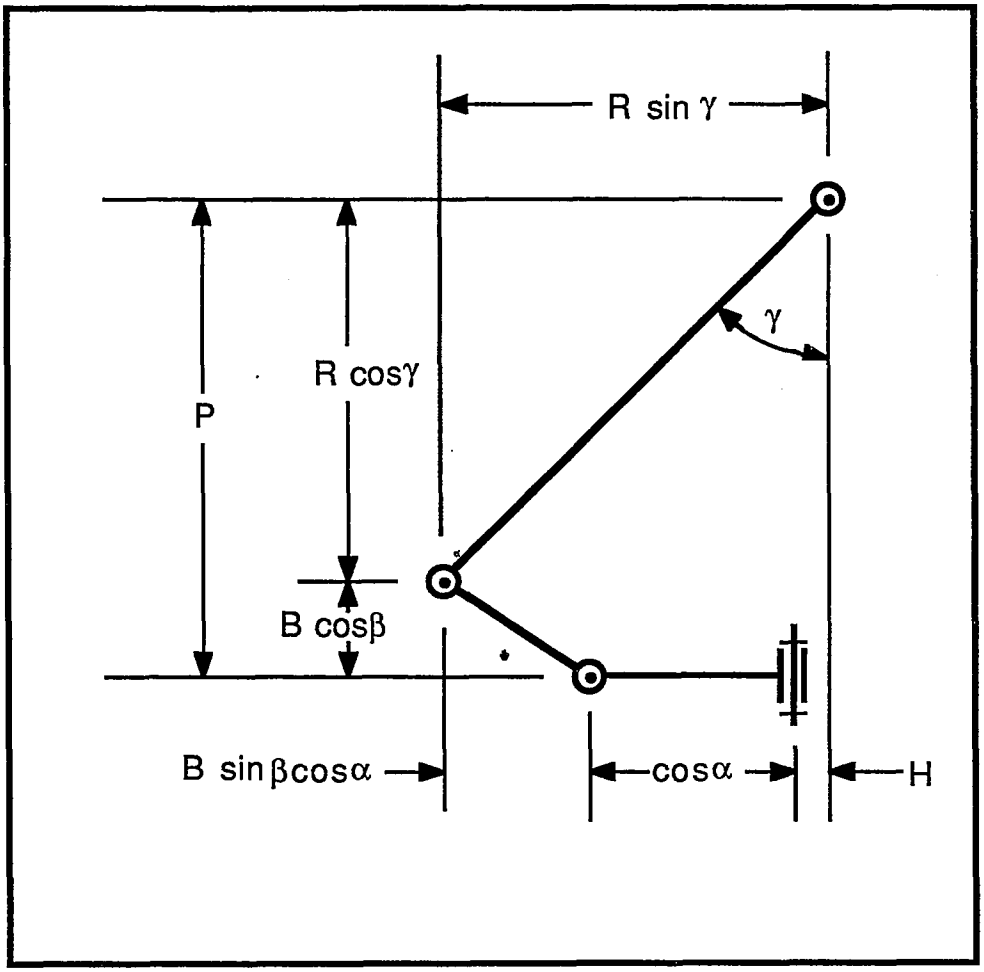


Figure 7: Projection of RRSC onto XZ Plane

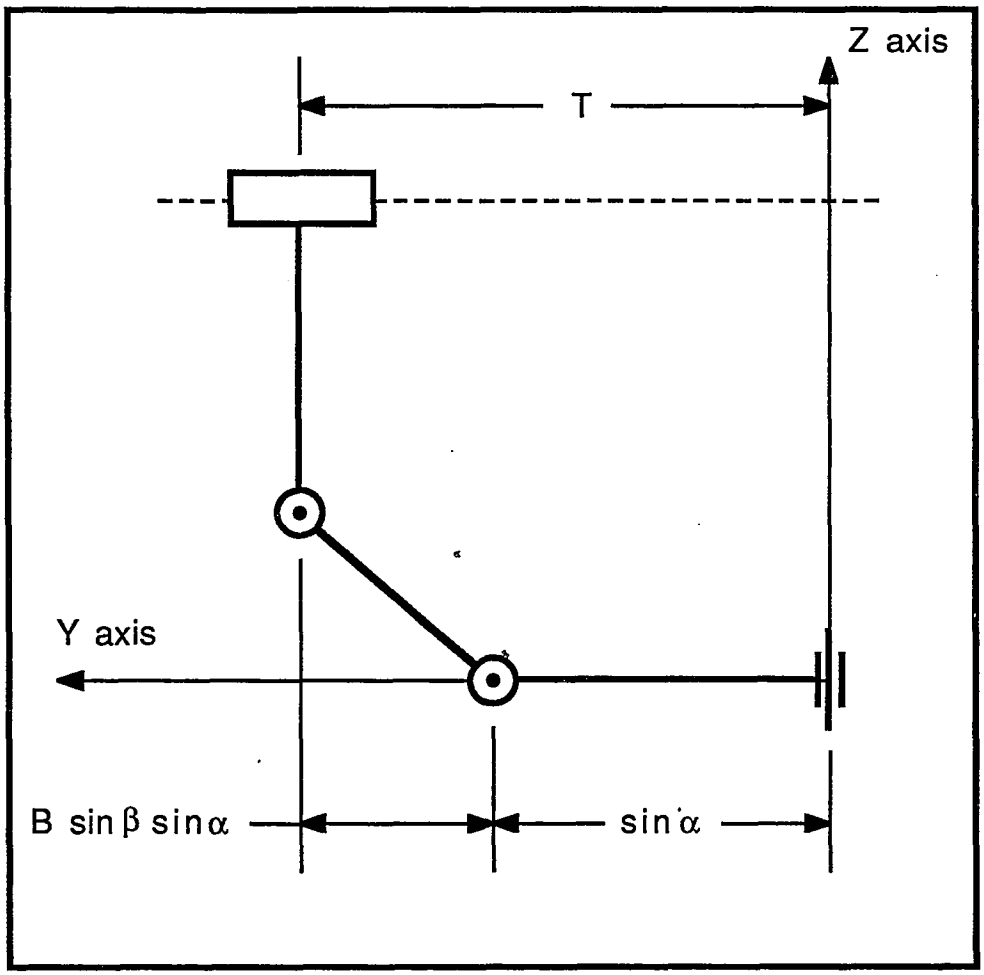


Figure 8: Projection of RRSC onto YZ Plane

3.1.1 Quadruple Dwell This is the most constrained of the RRSC configurations; the constraints are reiterated here:

$$R^2 = (1 + H)^2 + (P - B)^2 \quad (1)$$

$$H = 0 \quad (2)$$

$$BP/(R - P) = (P^2 + 1)^{0.5} \quad (3)$$

Since H = 0, a subspace of the original (R, P, B, H) linkage space will be used, namely (R, P, B). This reduces our linkage space; a parameter space must now be found to map into the subspace. Again, Hunt and Shrivastava have found a single parameter to describe this linkage [18]. This parameter, ψ, is half the total span of the angular output, as shown in Figure 9. The values of R and P are explicit functions of ψ, given as:

$$R = \csc \psi \quad (7)$$

$$P = \cot \psi + B \quad (8),$$

and B may be found from the solution to the cubic equation:

$$X^3 - (1 - 2 \cos \psi) X^2 + (1 - \cos \psi + \cos^2 \psi) X - (1 - \cos \psi) / 2 = 0 \quad (9),$$

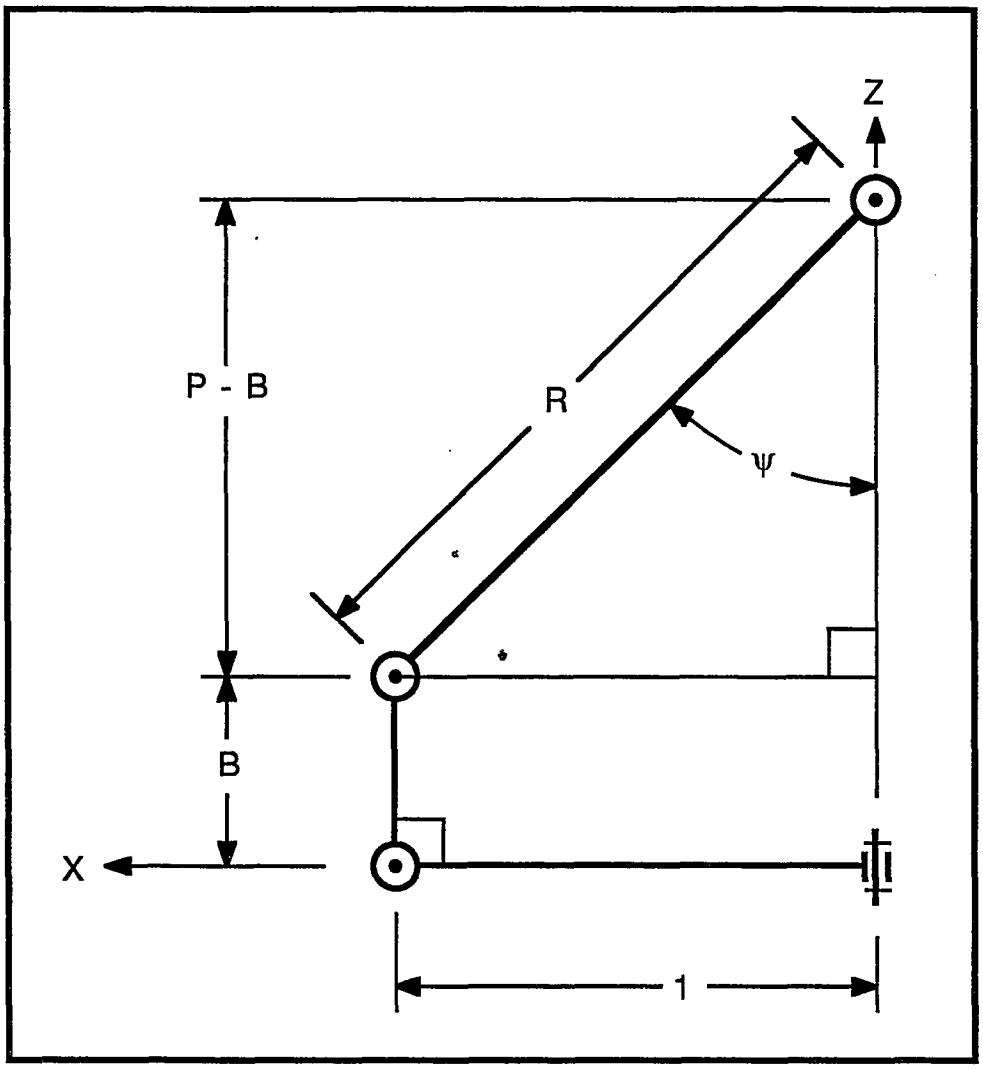


Figure 9: Projection of RRSC.4d at Crank Angle 0° onto XZ Plane

where

$$X = B \sin \psi \tag{10}$$

A further useful equation given in [18] is the relationship between the translational half-span T_{max}, and the link lengths:

$$T_{max} = R/P \tag{11}$$

To assure crank rotability, only one inequality constraint must be considered. This is the constraint assuring that the links can be assembled in certain critical positions, namely at crank angles 90° and 270°. In these positions the position of the spherical joint reaches its minimum Z value; as illustrated in Figure 10, the inequality

$$B \geq R - P \tag{12}$$

will ensure that the linkage will assemble at this point, thus also assuring full crank rotation. Inequality (12) is valid for all of the RRSC dwell configurations where $H = 0$ is true. In the RRSC.4d configuration, this inequality constraint is always satisfied; this may be demonstrated by considering Equation (3), rewritten as

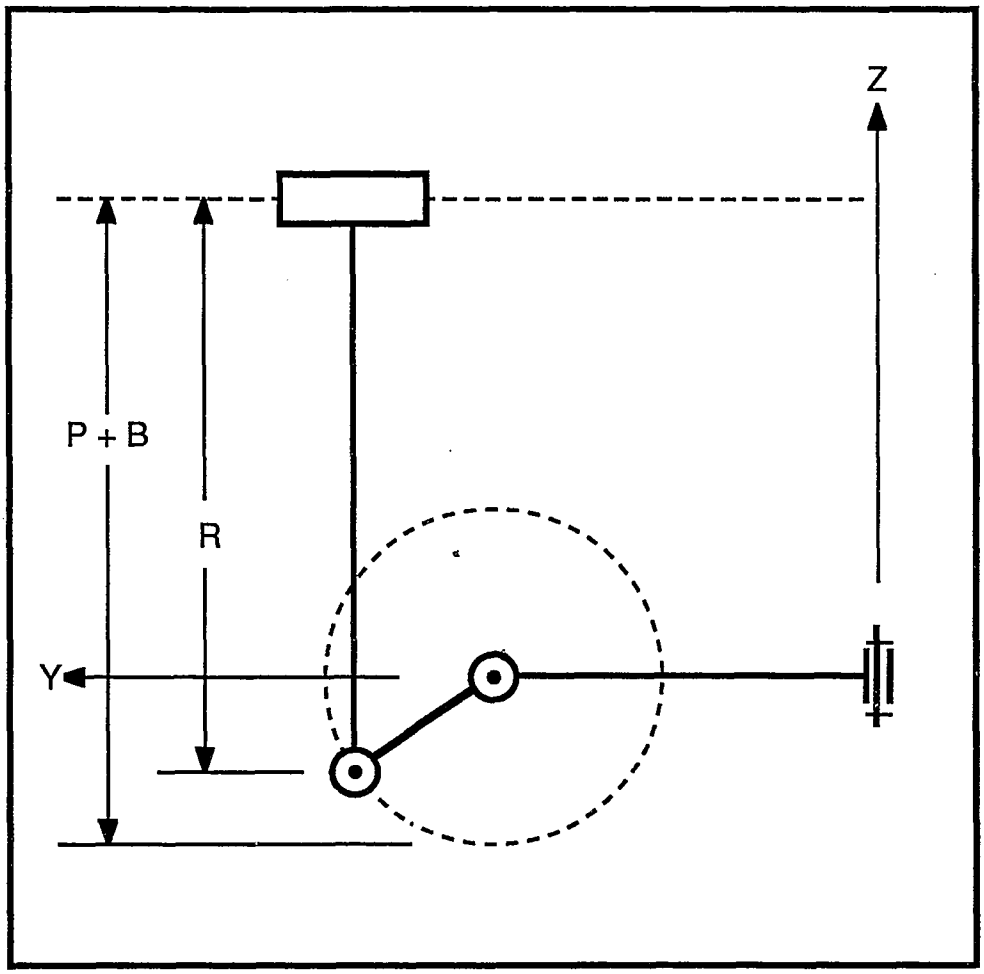


Figure 10: Projection of RRSC.4d at crank angle 90° onto YZ Plane, showing Assembly Requirement

$$B = (R - P)(P^2 + 1) / P = K(R - P) \qquad (13),$$

where

$$K = (P^2 + 1) / P > 1 \qquad (14)$$

Thus for the RRSC.4d configuration, and for all RRSC dwell configurations where Equations (2) and (3) hold, the crank is fully rotatable for all particular linkages.

To find some particular RRSC.4d linkages, a TK Solver model was made, using equations (7) through (11). The particular linkage dimensions for several values of ψ were determined automatically by the TK Solver program. These are given graphically in Figure 11, and agree with Hunt and Shrivastava's results.

SpaceLinks was then used to determine the general shape of the output functions. A matrix transformation model was developed for the RRSC configuration, and is given in Appendix C. A typical RRSC.4d linkage ($\psi = 45°$) was analyzed, and the translational and angular outputs, normalized with respect to their half-spans, were graphed. These are shown in Figure 12, where the dwell points are readily visible at crank angles 0°, 90°, 180°, and 270°.

The final results for the RRSC.4d linkage configuration are given in Figures 13 and 14; these are the graphs relating the angular half-span to the dwell range, for dwell accuracies of one and five percent. These were produced using SpaceLinks in batch

Figure 11: Link Length Curves for RRSC.4d

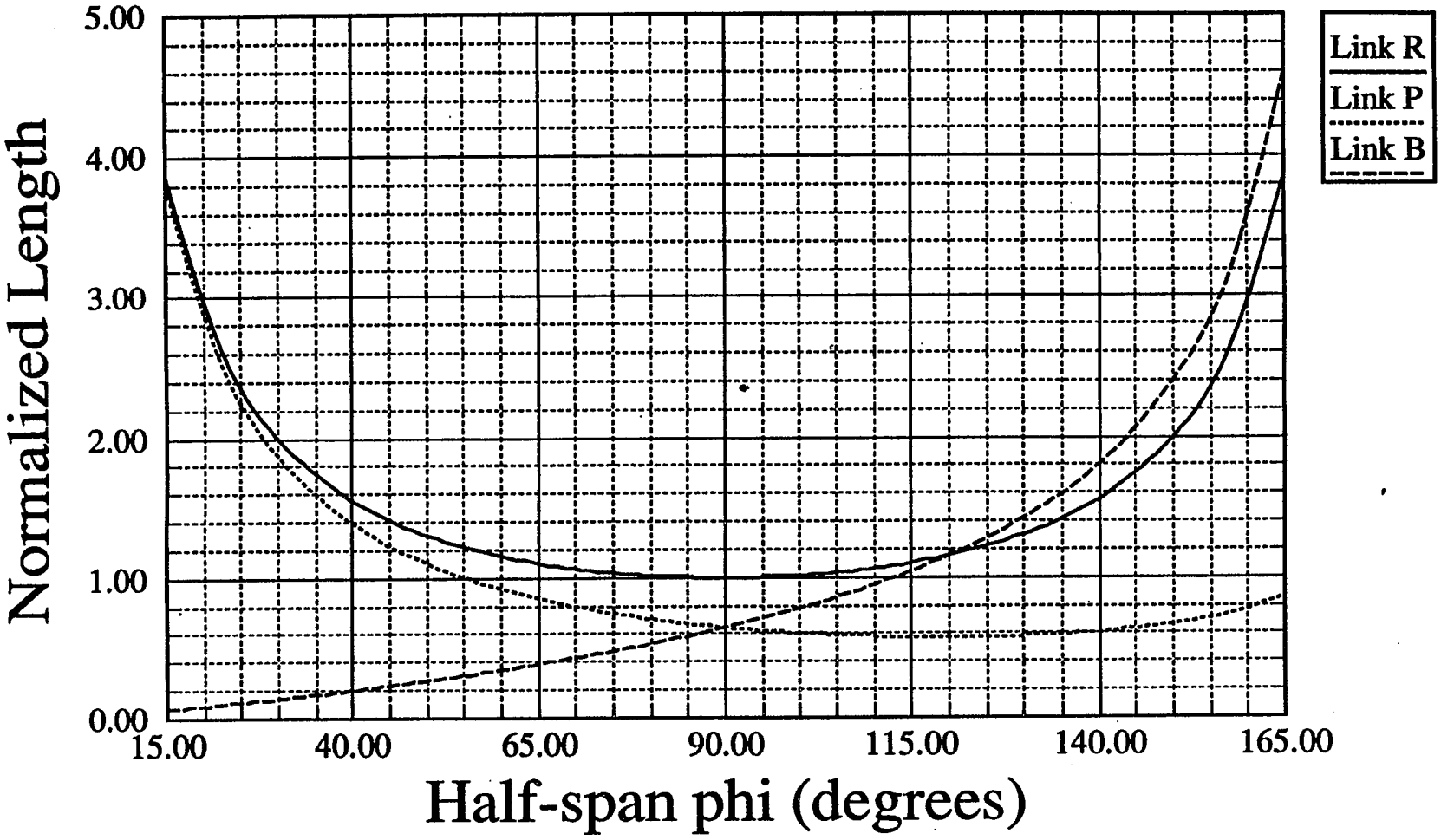


Figure 12: Cylinder Joint Output for RRSC.4d

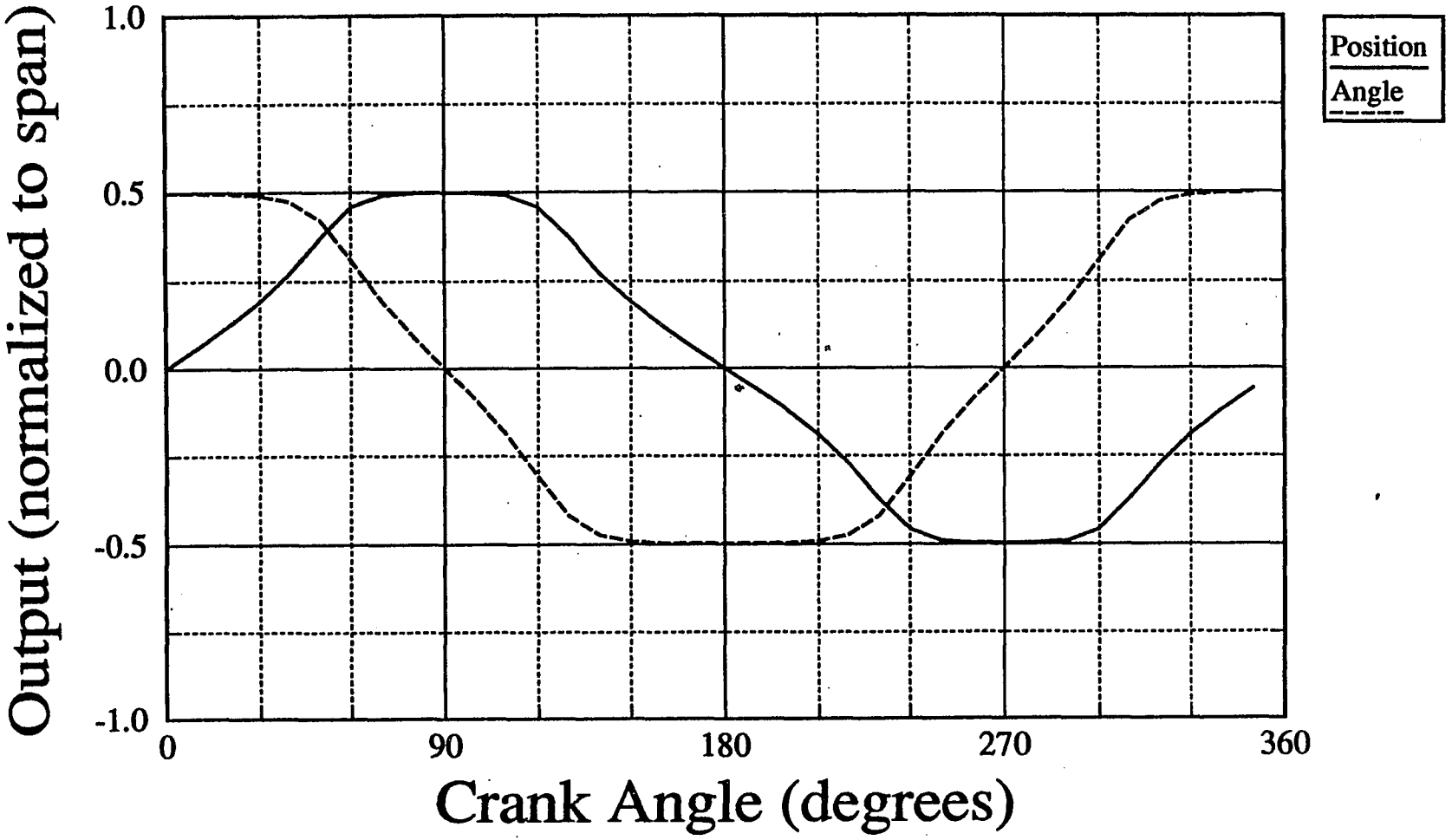


Figure 13: Angular Dwell Ranges for RRSC.4d

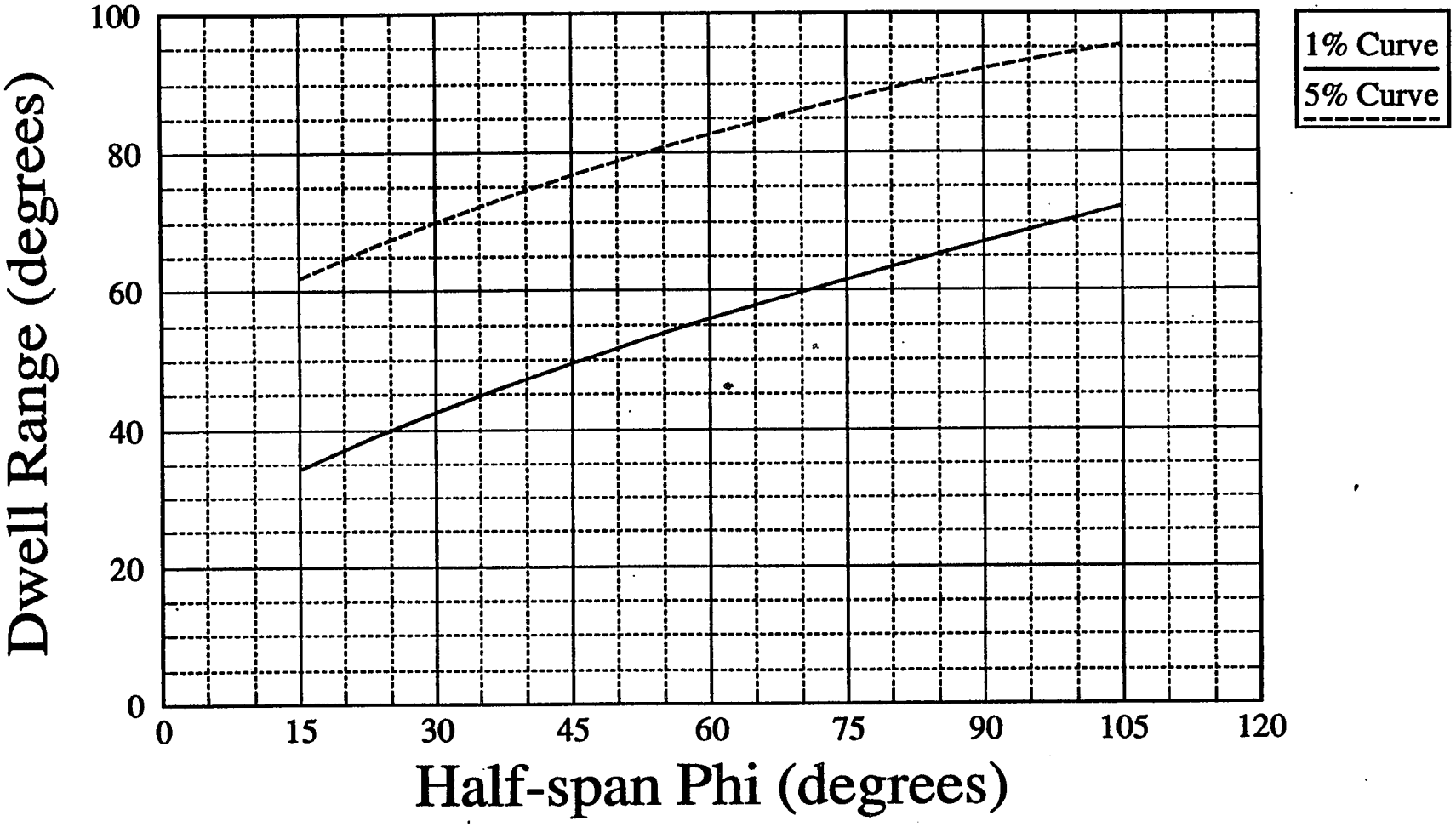
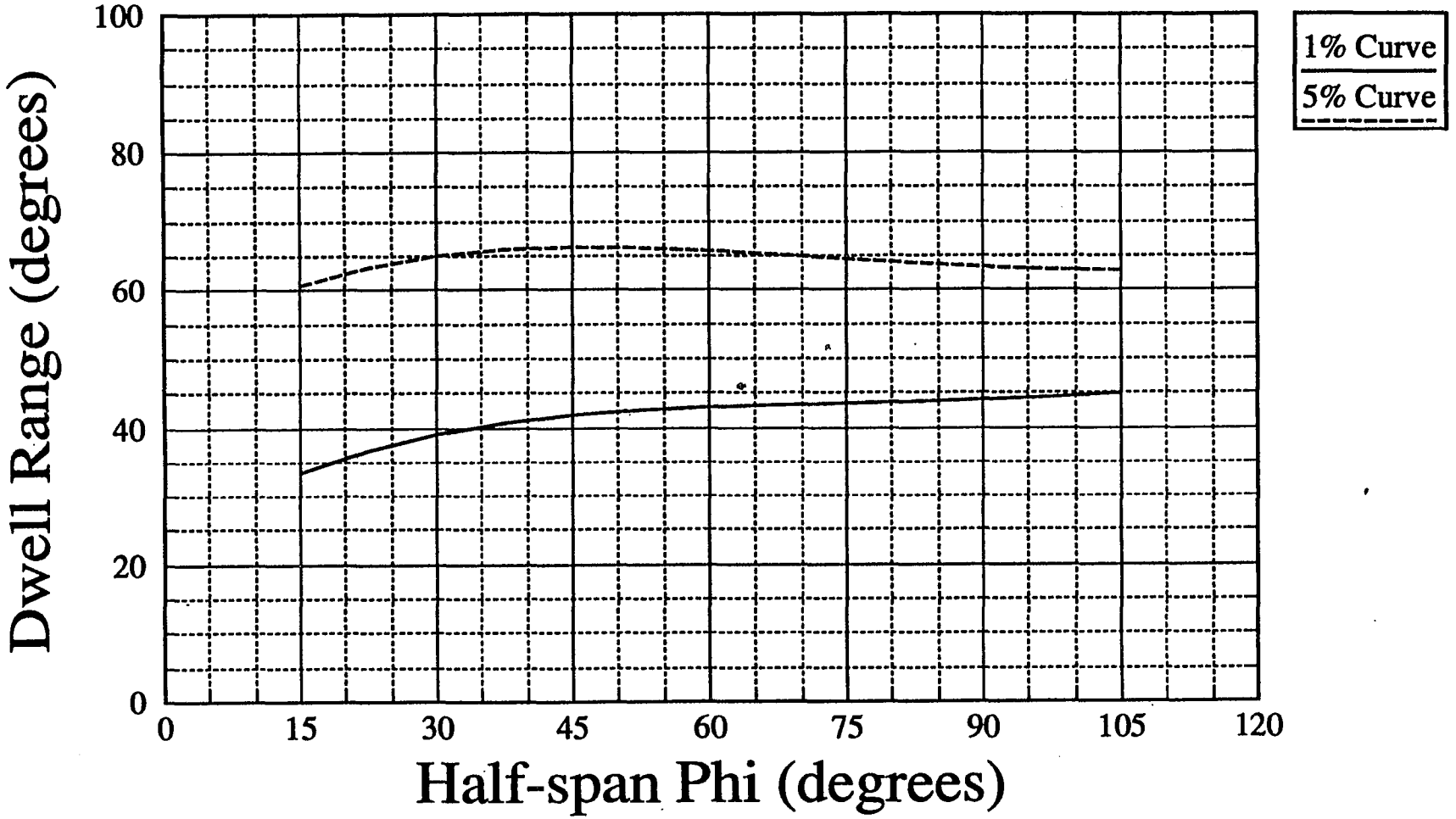


Figure 14: Translational Dwell Ranges for RRSC.4d



mode, using the results from the TK Solver link length model, but several new equations were added to find the dwell range and the dwell accuracy (or tolerance). For the angular output:

$$\text{range} = 2\,\alpha \tag{15}$$

$$\gamma = (1 - 2\ \text{tolerance})\,\psi \tag{16}$$

For the translational output:

$$\text{range} = 2\,(90^\circ - \alpha) \tag{17}$$

$$T = (1 - 2\ \text{tolerance})\,T_{max} \tag{18}$$

These equations were solved for the range, given ψ (thus also the link lengths), and the tolerance required. These graphs may be used to design a double dwell linkage with respect to either of the output variables, or a quadruple dwell linkage.

3.1.2 Double Angular Dwell By removing Equation (3) as a constraint on RRSC.4d, the translational dwell condition may be removed. This produces the

RRSC.2ad configuration, which has two identical dwells at the extrema of the angular output function, separated by a crank angle of 180°. The dwell conditions are reiterated here:

$$R^2 = (1 + H)^2 + (P - B)^2 \quad (1)$$

$$H = 0 \quad (2)$$

Again, H = 0 and so the (R, P, B) subspace may be used; this removes one variable and one constraint. This configuration may be described by two parameters, as there are three remaining design variables and one remaining equality constraint. Since this configuration produces angular dwell and is similar to the RRSC.4d configuration, one sensible choice for a parameter would be the angular half-span used in the RRSC.4d configuration. Before this parameter can be used, however, a demonstration must be made of its validity in this new and less constrained case.

For this configuration, Equation (1) may be rewritten as:

$$R^2 = 1 + (P - B)^2 \quad (19)$$

This is a hyperbola in the variables R and (P-B); it also describes a right triangle where one leg is unity (or Q, which equals one), the other leg is P-B, and the hypotenuse is R. When the crank angle is zero, this relationship forces the R_{23} joint variable to be 90°, and the projection of link lengths onto the XZ-plane is as shown in

Figure 15. From the figure, the relationship between P-B and ψ is

$$P - B = (P - B) / Q = \cot \psi \qquad (20),$$

which is equivalent to Equation (8). An equation for R in terms of ψ may also be found from the figure; this equation is equivalent to Equation (7). The angular half-span ψ therefore may be used as one parameter. The other parameter was chosen to be the normalized link length B.

Since H = 0 for this configuration, crank rotability is equivalent to Inequality (12); since Equation (3) no longer holds, however, full rotation is not assured for all possible linkages, and the rotability constraint must be checked. This may be more conveniently accomplished in the parameter space than in the design (link length) space, so the inequality was converted to

$$B \geq (1 - \cos \psi) / 2 \sin \psi \qquad (21)$$

A TK Solver model was developed to study this configuration. Equations (4), (5), (7), (8), (14), and (15), and Inequality (21), were incorporated in the model, and the TK Solver program automatically solved for curves of constant dwell range. These curves were plotted along with the results of a C program which calculated B for the equality portion of (21); these plots are given in Figures 16 and 17.

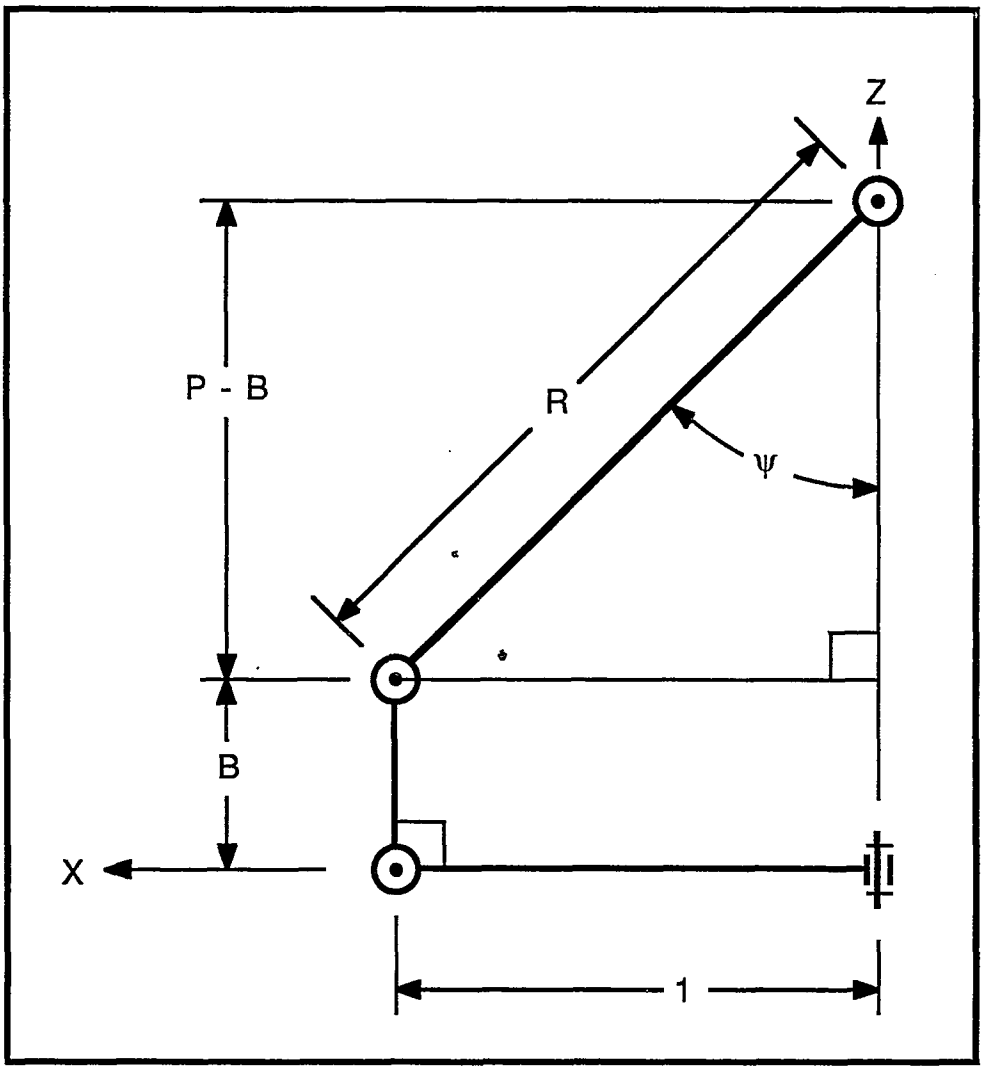


Figure 15: Projection of RRSC.2ad at Crank Angle 0° onto XZ Plane

Figure 16: 1% Dwell Ranges for RRSC.2ad

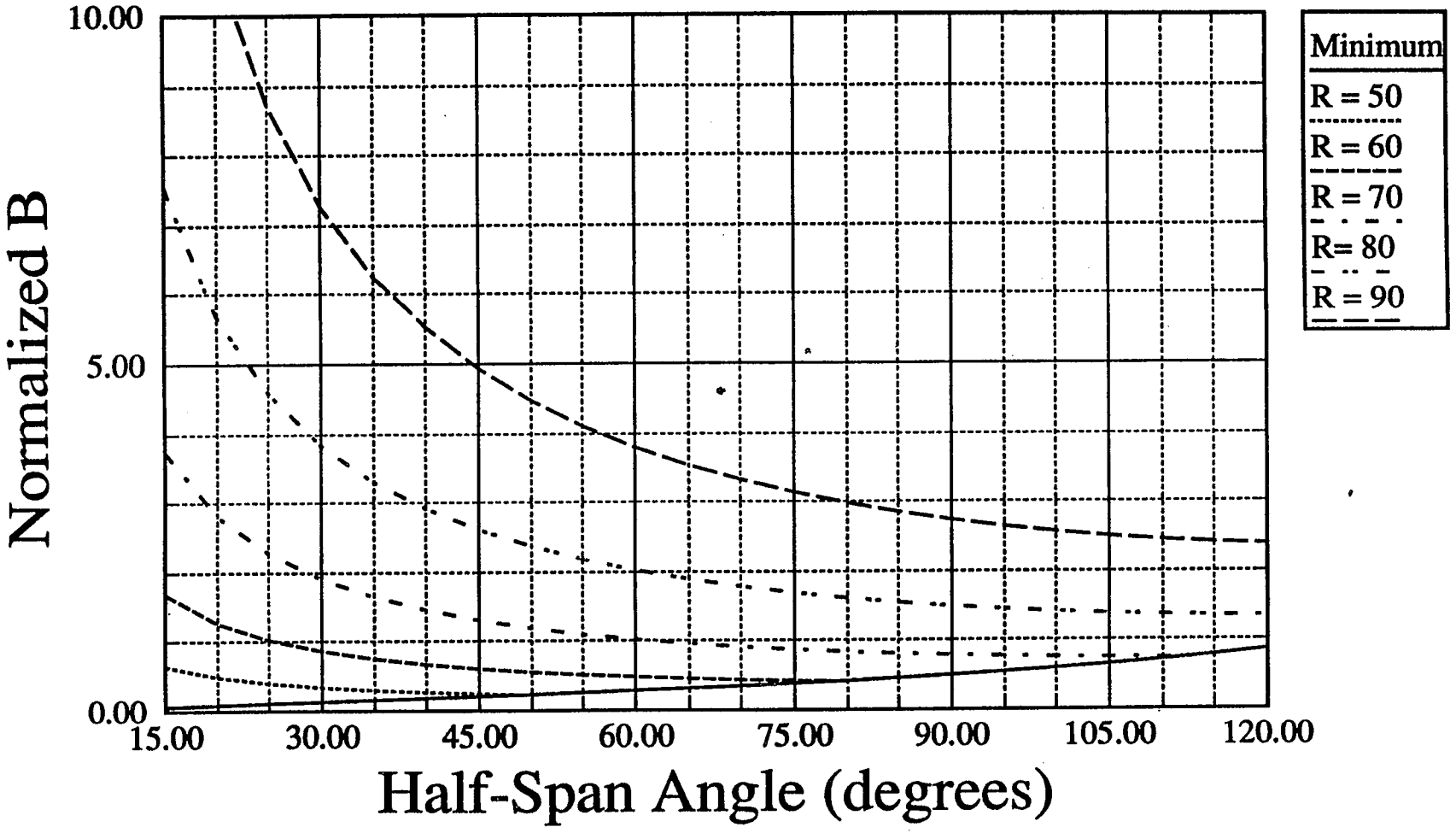
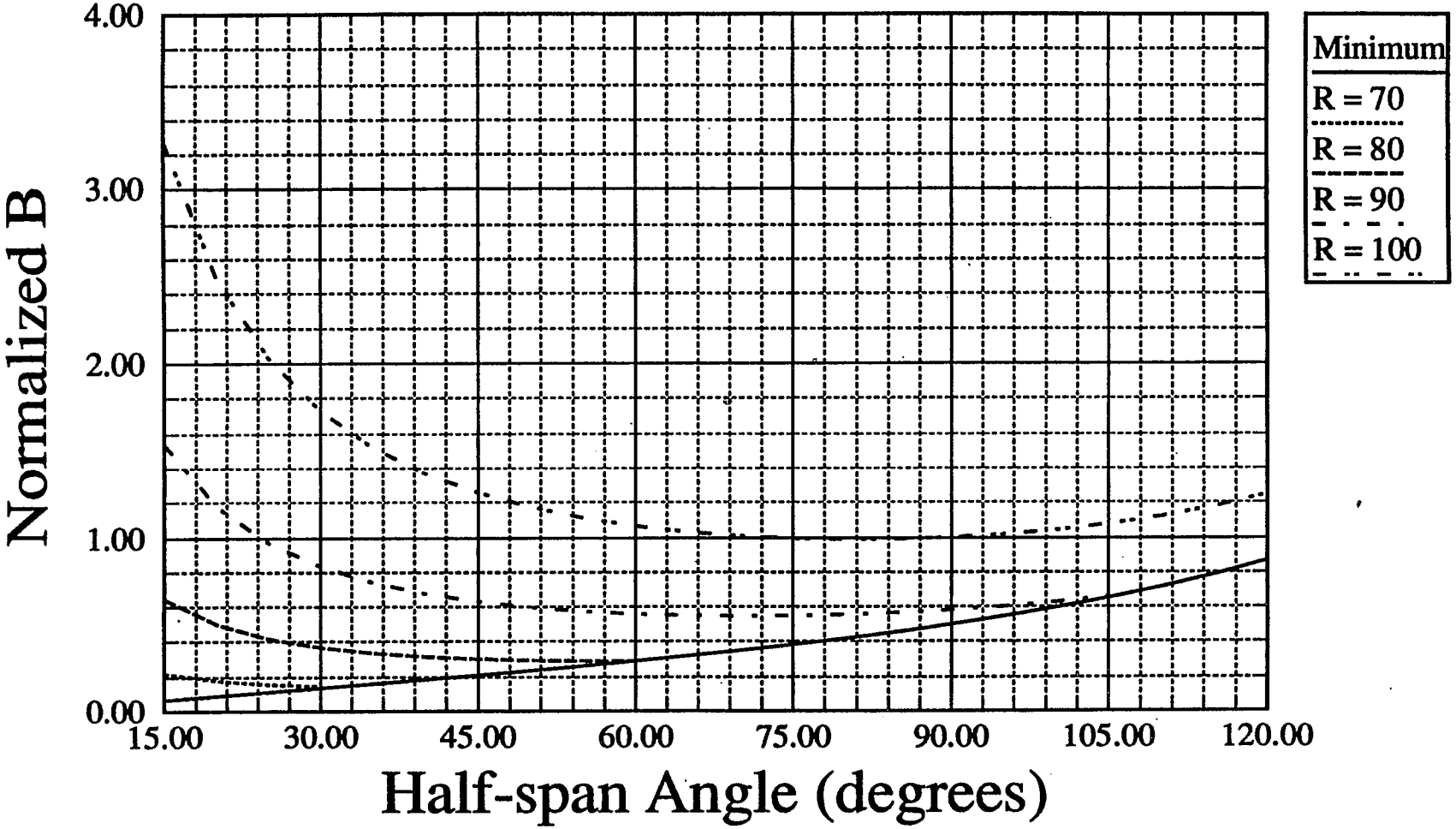


Figure 17: 5% Dwell Ranges for RRSC.2ad



3.1.3 Double Translational Dwell This linkage configuration has two equality constraints and four design variables. The equality constraints are shown again here:

$$H = 0 \tag{2}$$

$$BP/(R - P) = (P^2 + 1)^{0.5} \tag{3}$$

By considering only the (R, P, B) subspace, the number of variables may be reduced to three and the number of equality constraints to one, thus two parameters define the dwell curve through the design space. There is no need for a crank rotation constraint, as Equation (3) guarantees that Inequality (12) always holds.

As in the angular dwell case, the half-span may be used as one parameter; in the translational dwell case, however, the half-span is not ψ but T_{max}, the maximum normalized distance that the cylinder travels from the point of intersection of its axis with the R_{12} axis. For the quadruple dwell case, equation (11) was given by Hunt and Shrivastava to relate several link lengths with T_{max}. For the less restrictive RRSC.2td case, however, this equation must be shown to be valid, and other relationships derived to generate a parameter space and mapping function.

Consider Figure 18, where the RRSC.2td configuration, with a crank angle of 90°, the position of maximum translational displacement, is projected onto the YZ plane. The link length relationship

$$B^2 = (T - 1)^2 + (R - P)^2 \tag{22}$$

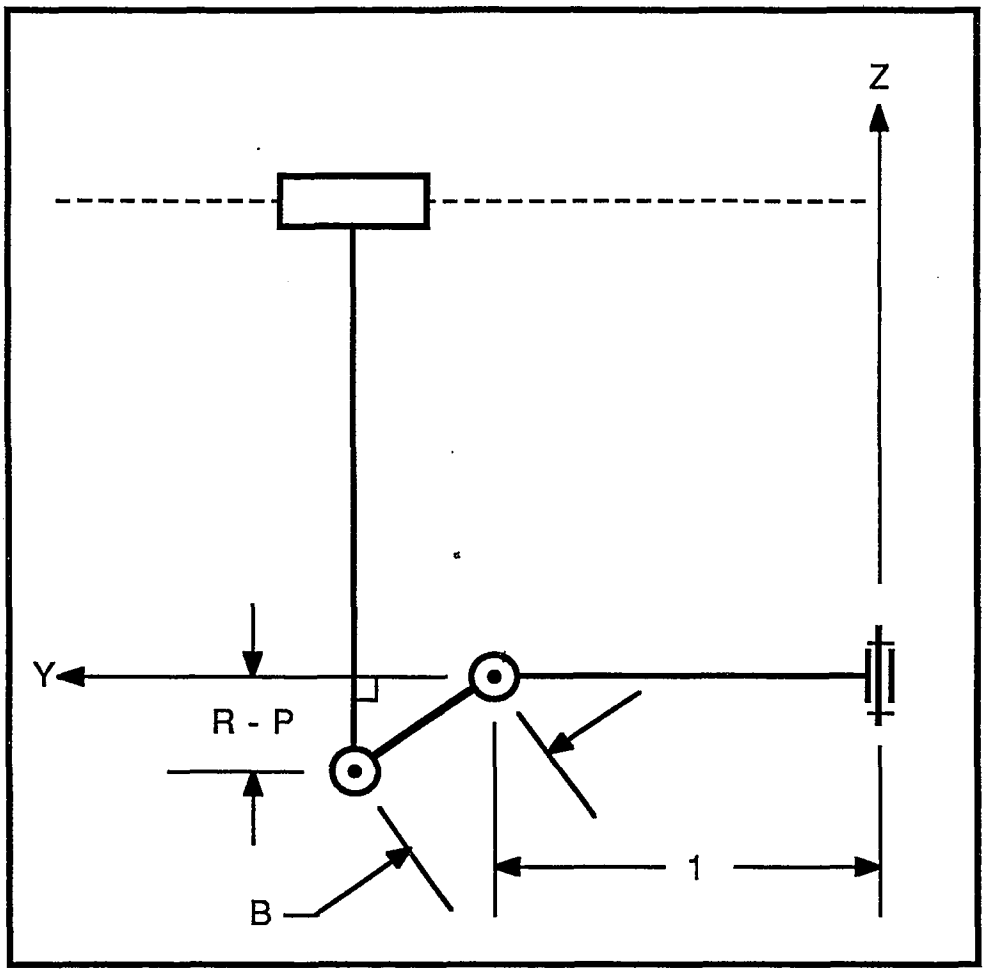


Figure 18: Projection of RRSC.2ad at Crank Angle 90°

may be combined with equation (3) to give

$$(T - 1)^2 + (R - P)^2 = (R - P)^2 (P^2 + 1) / P^2$$
$$= (R - P)^2 + (R - P)^2 / P^2 \qquad (23)$$

which may be reduced to

$$T - 1 = (R - P) / P = R/P - 1 \qquad (24)$$

which is equivalent to equation (11). Combining equation (24) with equation (3) gives

$$B = (T - 1) (P^2 + 1)^{0.5} \qquad (25)$$

thus, given T as a parameter, a logical choice for the other parameter would be P; then R could be given as

$$R = P T \qquad (26)$$

and the mapping function is complete.

Equations (25) and (26) were incorporated into a TK Solver program with equations (4) - (6) and (17) - (18), and curves of constant dwell range were automatically solved for, to produce the graphs in Figures 19 and 20.

3.1.4 Single Angular Dwell This configuration has only one equality constraint, so it should have three parameters. Since H is not necessarily zero in this case, the full (R, P, B, H) design space must be used; further, ψ as it was previously defined will no longer serve as the angular half-span or as a parameter. This configuration requires a new approach.

In this configuration, the only constraint required to produce the dwell is equation (1). This produces the right triangle shown in Figure 21 when crank angle α is zero. Thus the spherical pair for all RRSC.1ad linkages must pass through the point (1, 0, B).

For this linkage to be able to rotate, two conditions must be met. In Figure 22, the two circles are the section cut by the XZ plane through the reachable space of the $R_{12}R_{23}$ open chain (a torus); the $C_{14}S_{43}$ open chain is represented by an arc that must pass through the top of the leftmost circle (point M) to satisfy the dwell condition. The first condition for crank rotatability is that the arc must intersect the rightmost circle (point N); the second condition is that no point on the arc, between points M and N, may lie outside of the region defined by

$$|Z| < B \qquad (27)$$

Figure 19: 1% Dwell Ranges for RRSC.2td

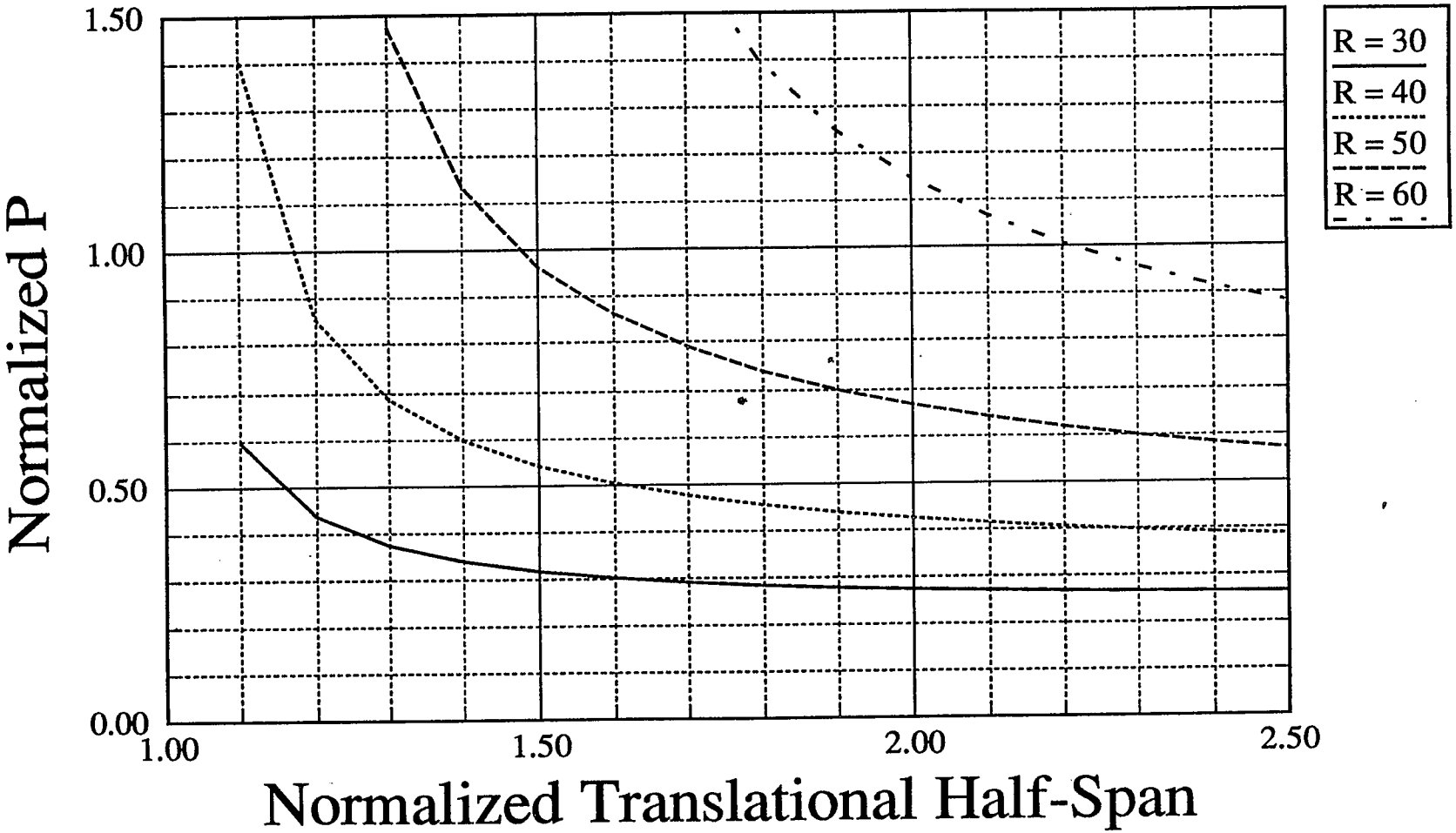
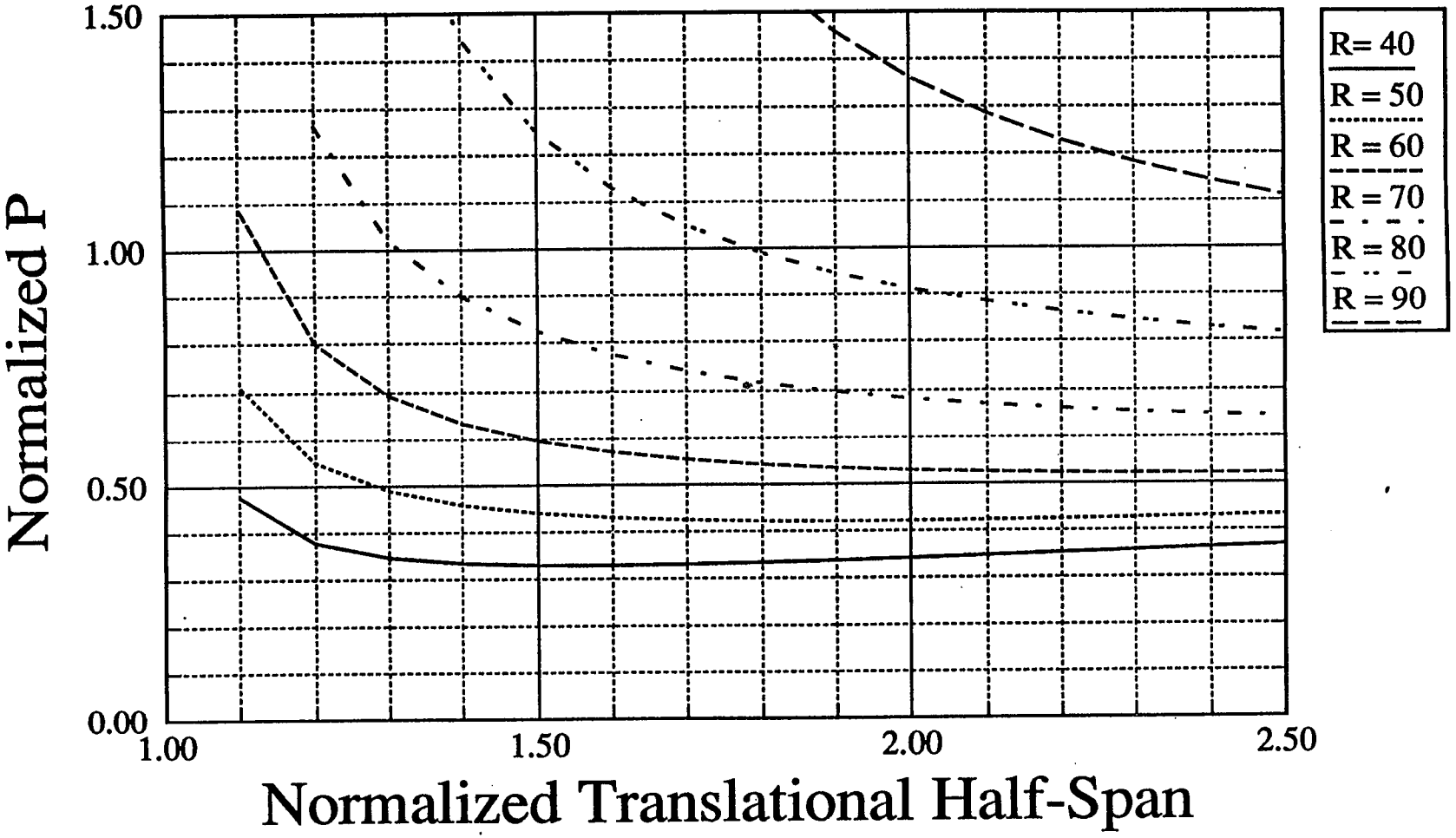


Figure 20: 5% Dwell Ranges for RRSC.2td



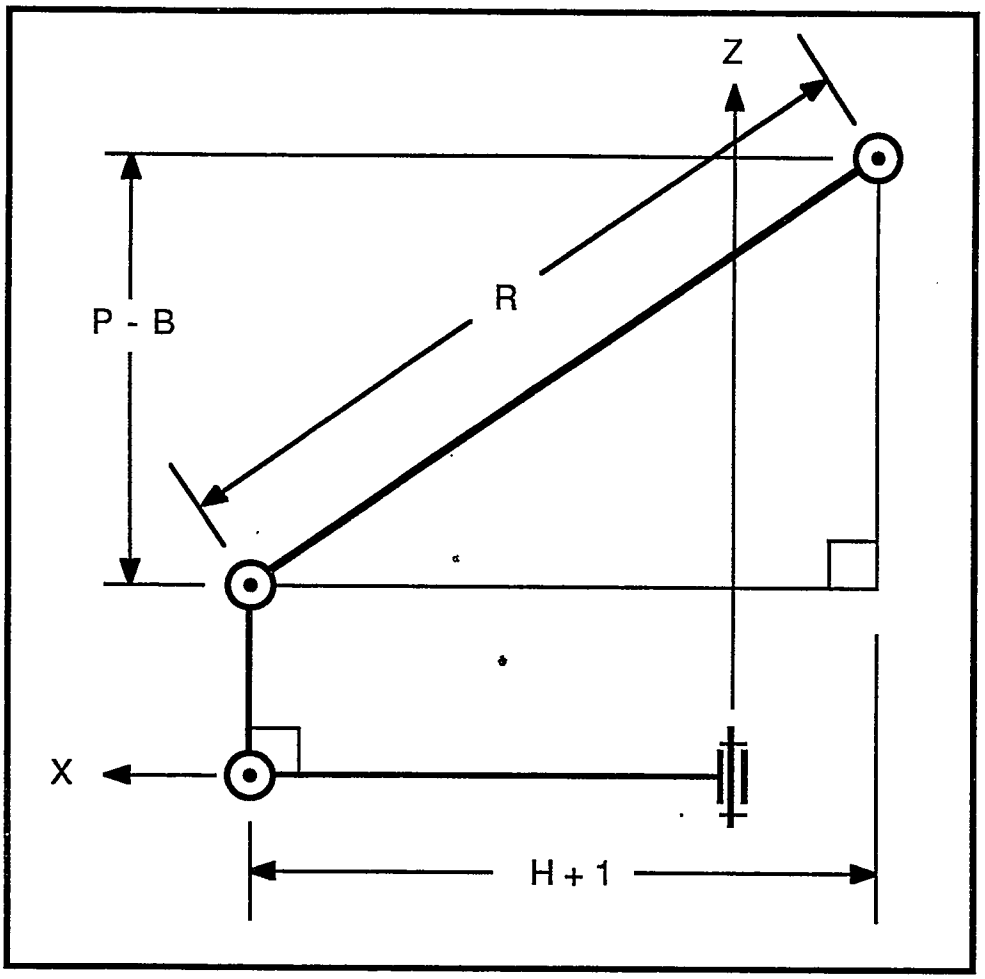


Figure 21: Projection of RRSC.1ad at Crank angle 90° onto XZ Plane

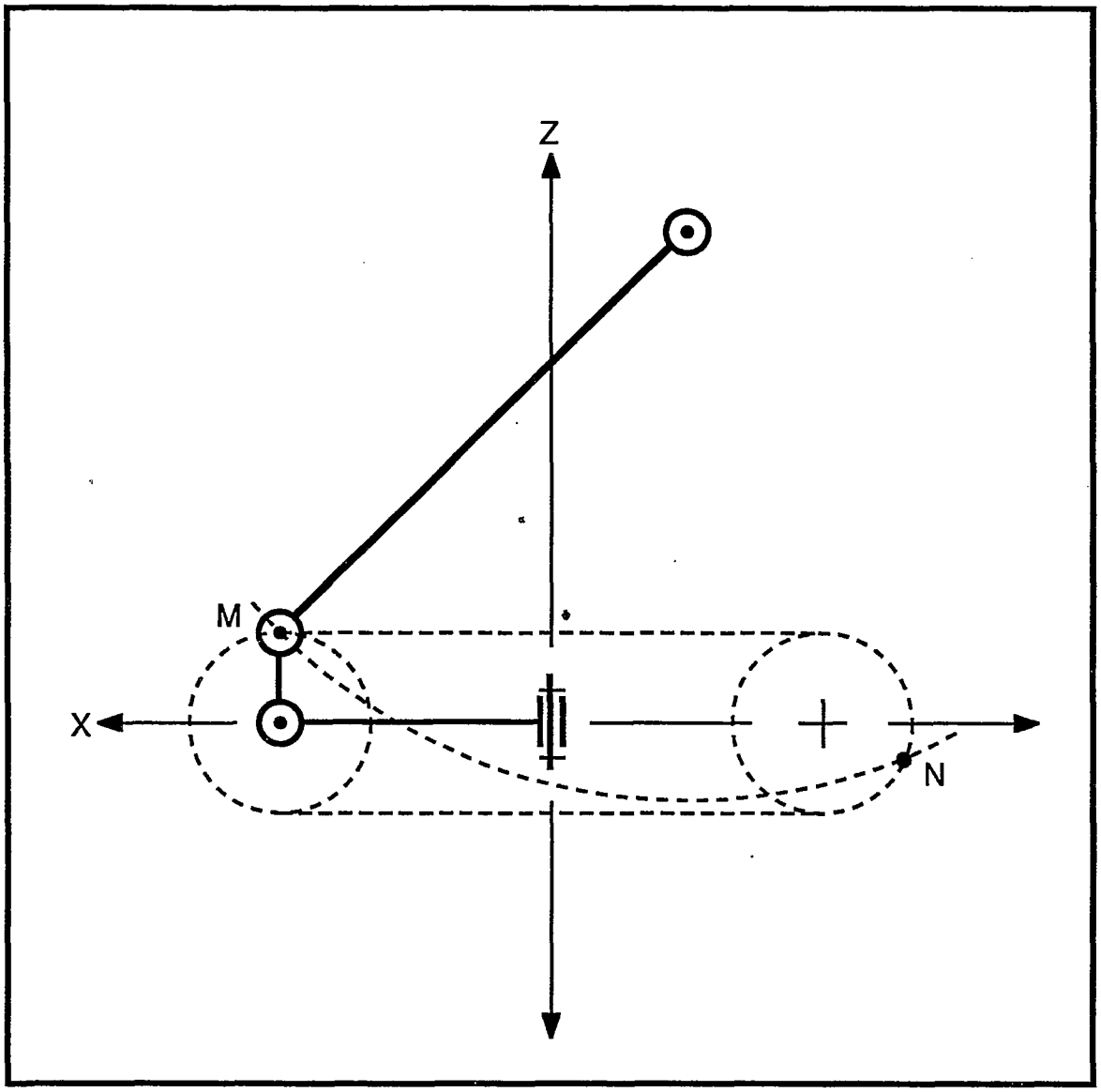


Figure 22: Intersection of CS Arc with RR Circles for RRSC.1ad

This last condition may be satisfied by either inequality (12) or

$$H > 1 \tag{28}$$

The first condition for crank rotatability is the motivation for the choice of parameters in the RRSC.1ad linkage. As shown in Figure 23, the point N on the rightmost circle may be defined by the radius B and an angle θ, measured from the top of the circle, and the fixed pivot for the arc must lie on the perpendicular bisector of the line segment between M and N. This perpendicular bisector is a line that may be represented as a point and a scalar multiple (τ) of a two dimensional vector. The point chosen was the midpoint of the line segment, and the vector chosen was the vector from the midpoint to point M, rotated 90°. The equations for the fixed pivot position may be written as:

$$X = B\,(t \cos q - t - \sin q) / 2 \tag{29}$$

$$Z = B\,(1 + \cos\theta + \tau \sin\theta) / 2 + \tau \tag{30}$$

Thus B, θ, and τ may be used as design parameters:

$$H = B\,(\sin\theta + \tau - \tau \cos\theta) / 2 \tag{31}$$

$$P = B\,(1 + \cos\theta + \tau \sin\theta) / 2 + \tau \tag{32}$$

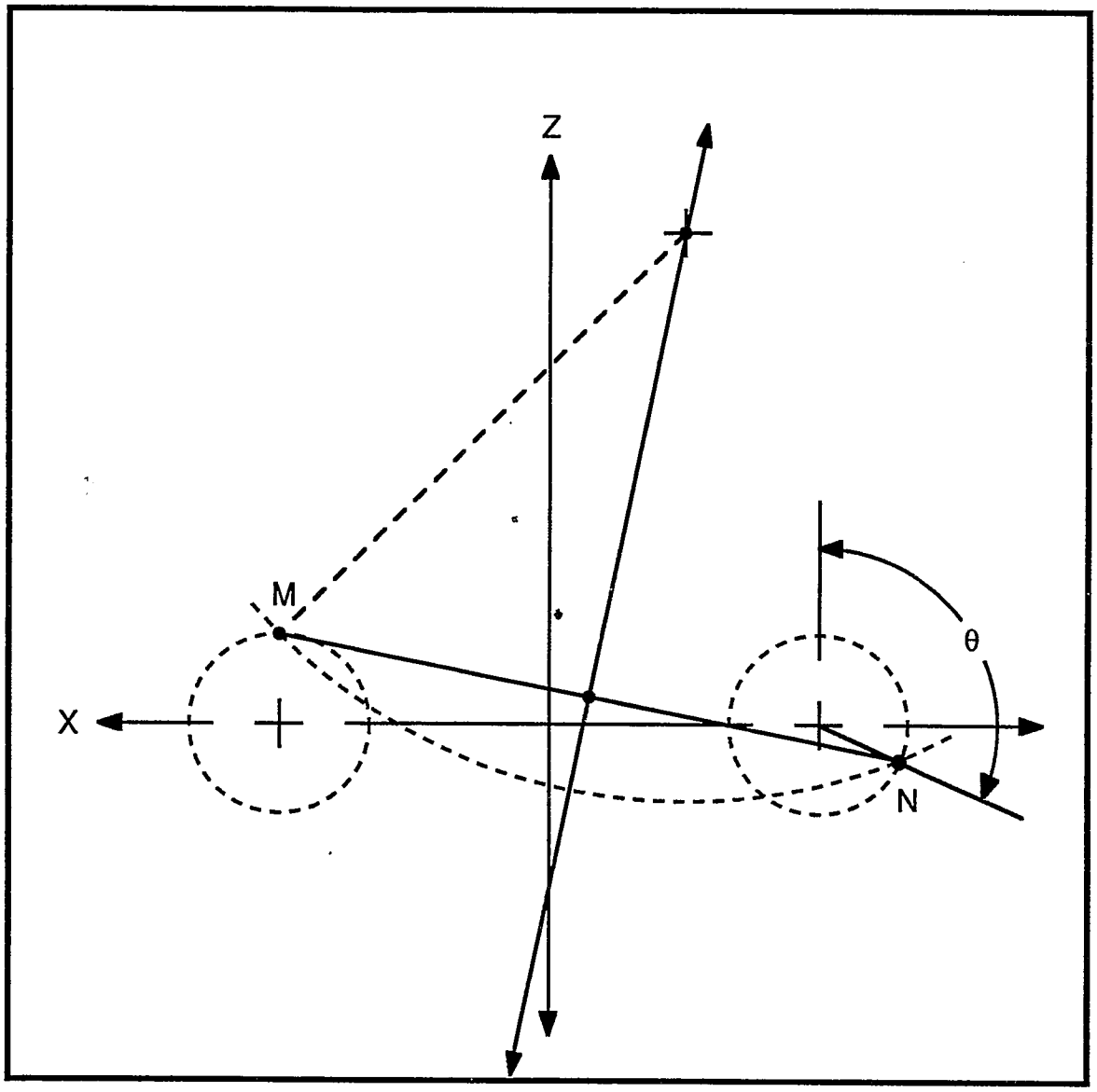


Figure 23: Angle θ and Perpendicular Bisector for RRSC.1ad

and R can be found by substituting equations (31) and (32) into equation (1):

$$R = [\ (1 + B\ (\sin\theta + \tau - \tau\cos\theta)\ /\ 2)^2 + (B\ (1 + \cos\theta + \tau\sin\theta)\ /\ 2 + \tau - B)^2\ ]^{0.5} \qquad (33)$$

Rotatability constraints will be left in the link length forms of inequalities (12) and (28) for convenience.

The new parameters θ and τ also have limiting values placed on them. Since H and P are both greater than zero, the intersection point must lie on the major arc of the rightmost circle between the points of intersection of the circle with the tangent lines passing through M, as shown in Figure 24. Thus the limits on θ are:

$$0 < \theta < 180^\circ + 2\tan^{-1}(B/2) \qquad (34)$$

There are two possible intersection points between the arc and the rightmost circle; the intersection point defined by θ must be the rightmost of these two. This may be assured by making the pivot point be above and to the right of a line through N and the center of the circle, as shown in Figure 25. This line intersects the perpendicular bisector as shown in the figure, and as long as inequality (34) holds, this intersection point will exist, and τ must be larger than or equal to the τ that defines this point on the perpendicular bisector. The value of τ for this point may be found by the simultaneous solution of the equations of the two lines, and is given by:

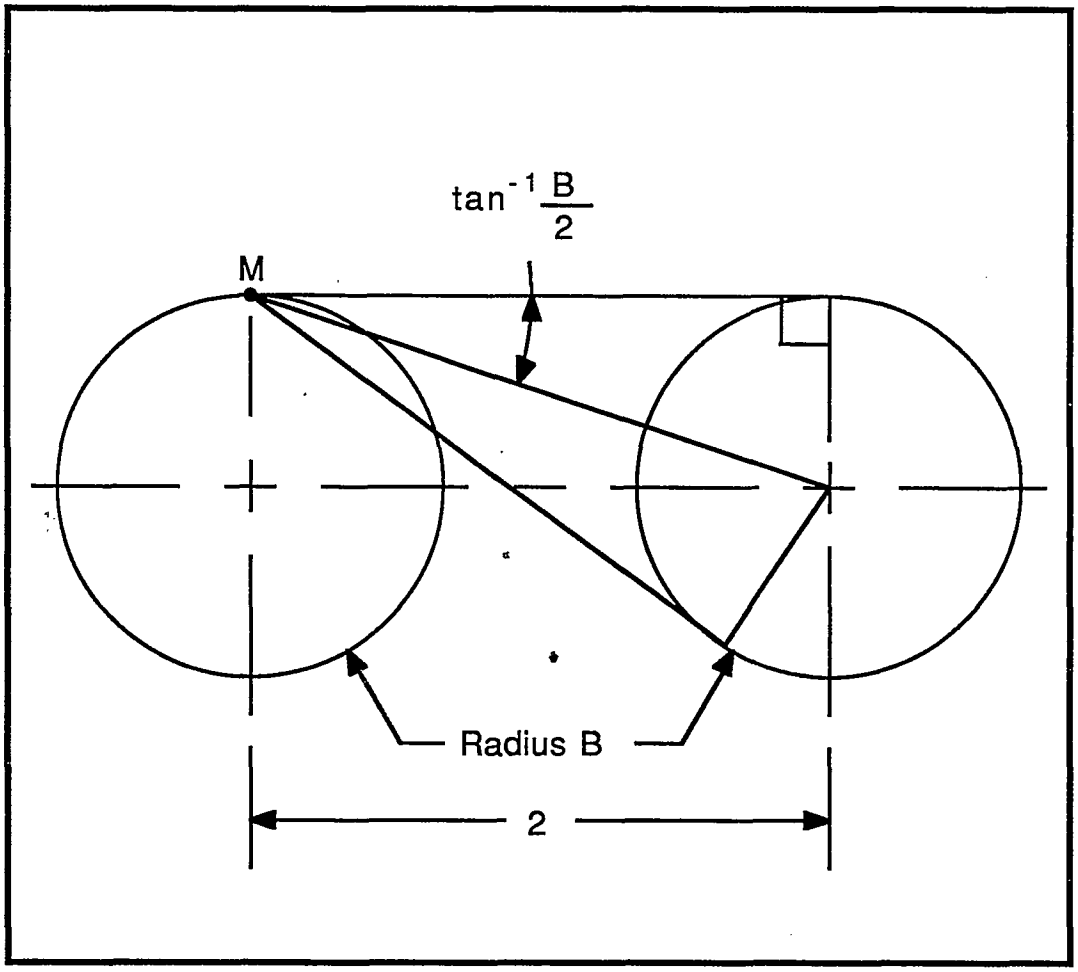


Figure 24: Tangents to Circle from Point M

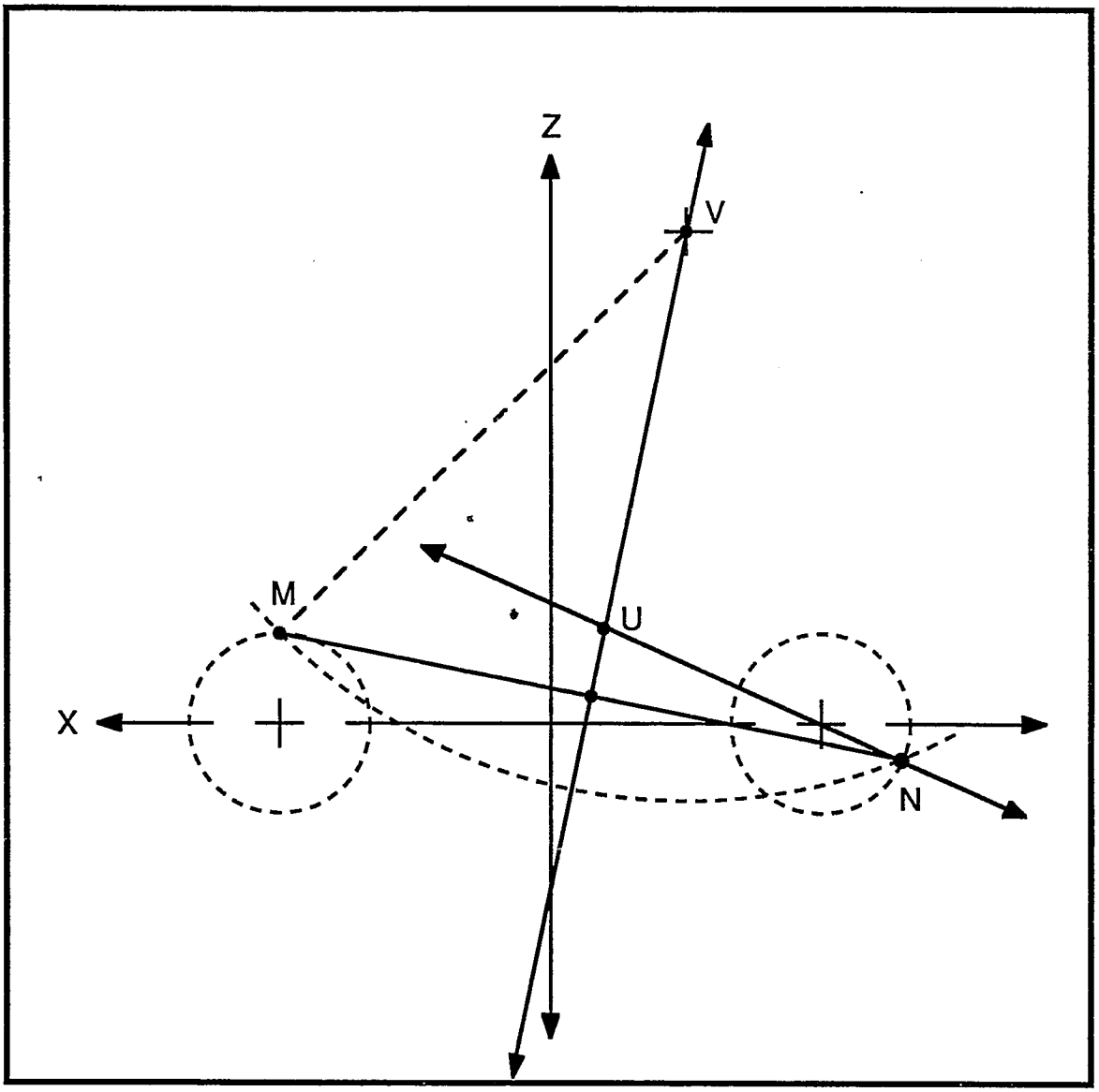


Figure 25: If Pivot V is Above Point U then N is Rightmost Intersection of Arc with Circle

$$\tau_{min} = [\cos\theta\,(B\sin\theta/2 - 1) - B\sin\theta\,(1+\cos\theta)/2\,]$$
$$\div [\,B\cos\theta\,(\cos\theta - 1)/2 + \sin\theta\,(1 + B\sin\theta/2)\,] \qquad (35)$$

The span may be found by considering the right triangle formed by M, the midpoint of MN, and the fixed pivot of the arc, as shown in Figure 26. Since the distance from the midpoint to the pivot is always τ times the distance from the midpoint to point M, the angle ψ shown in the figure, which is the half-span for RRSC.1ad is dependent solely on τ:

$$\psi = \cot\tau \qquad (36)$$

Equations (4) and (5) define the output function, and these were combined with equations (31) -(33) and (35) - (36), and inequalities (12), (28), and (34) in a TK Solver model; range and tolerance were represented by:

$$\text{range} = 2\,\alpha \qquad (37)$$

$$\text{tolerance} = (\mu - \gamma)\,/\,\text{span} = (\mu - \gamma)\,/\,2\,\psi \qquad (38)$$

These were solved, for different values of B, for curves of constant range, and these results were graphed as the design charts given in Figures 27 through 34.

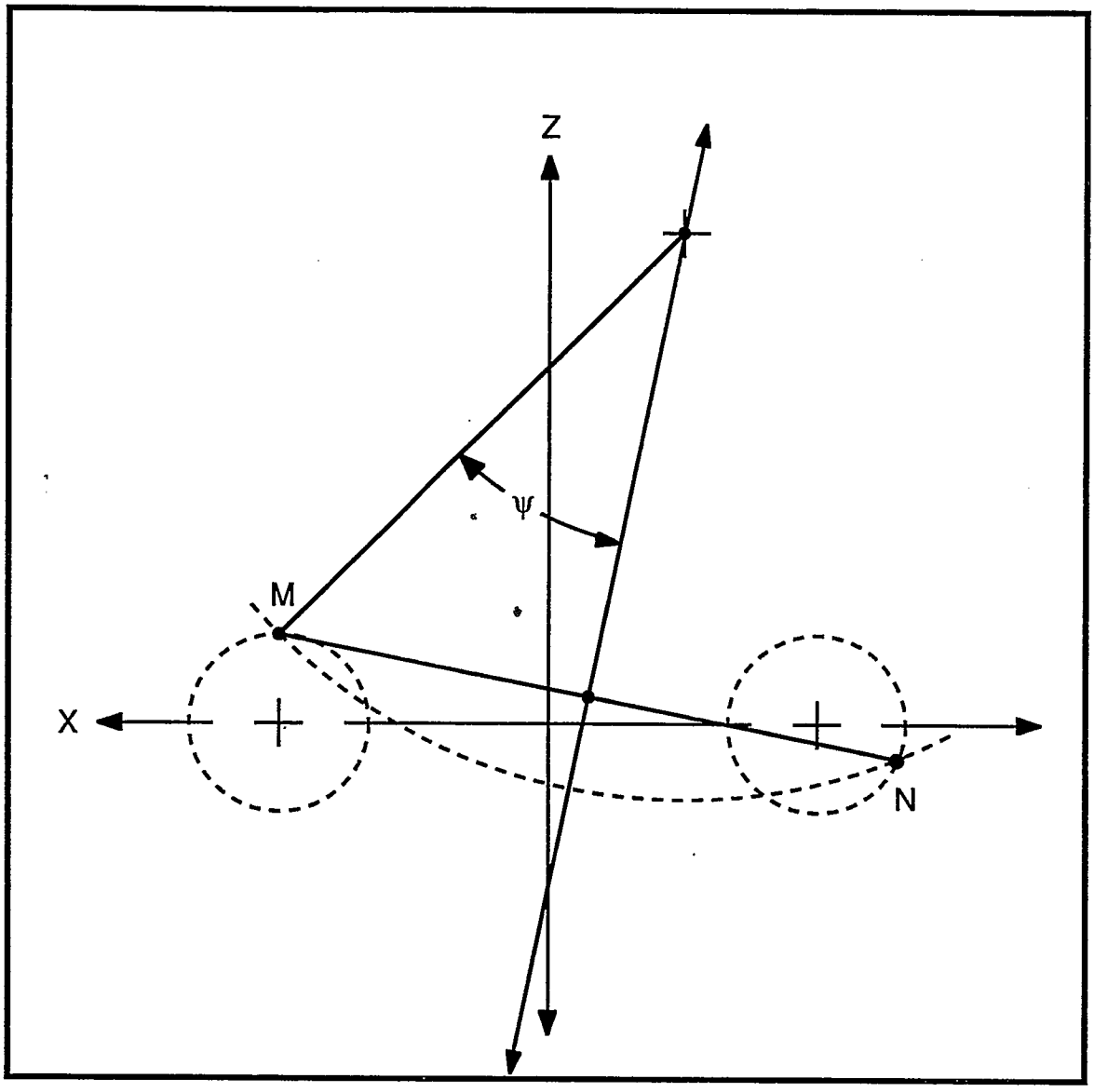


Figure 26: Half-span ψ for RRSC.1ad

Figure 27: 1% Dwell Ranges for RRSC.1ad (B = 0.5)

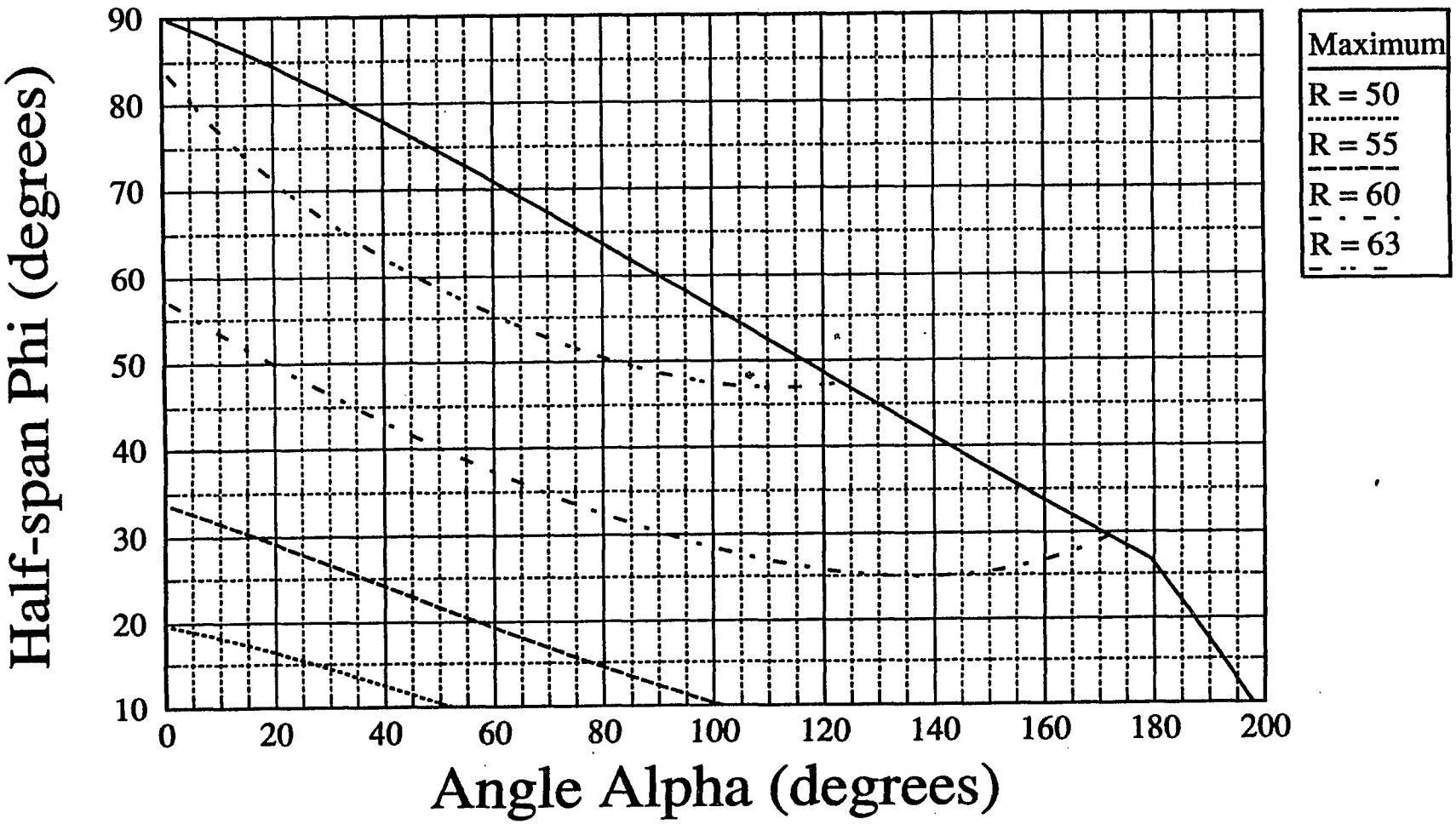


Figure 28: 5% Dwell Ranges for RRSC.1ad (B = 0.5)

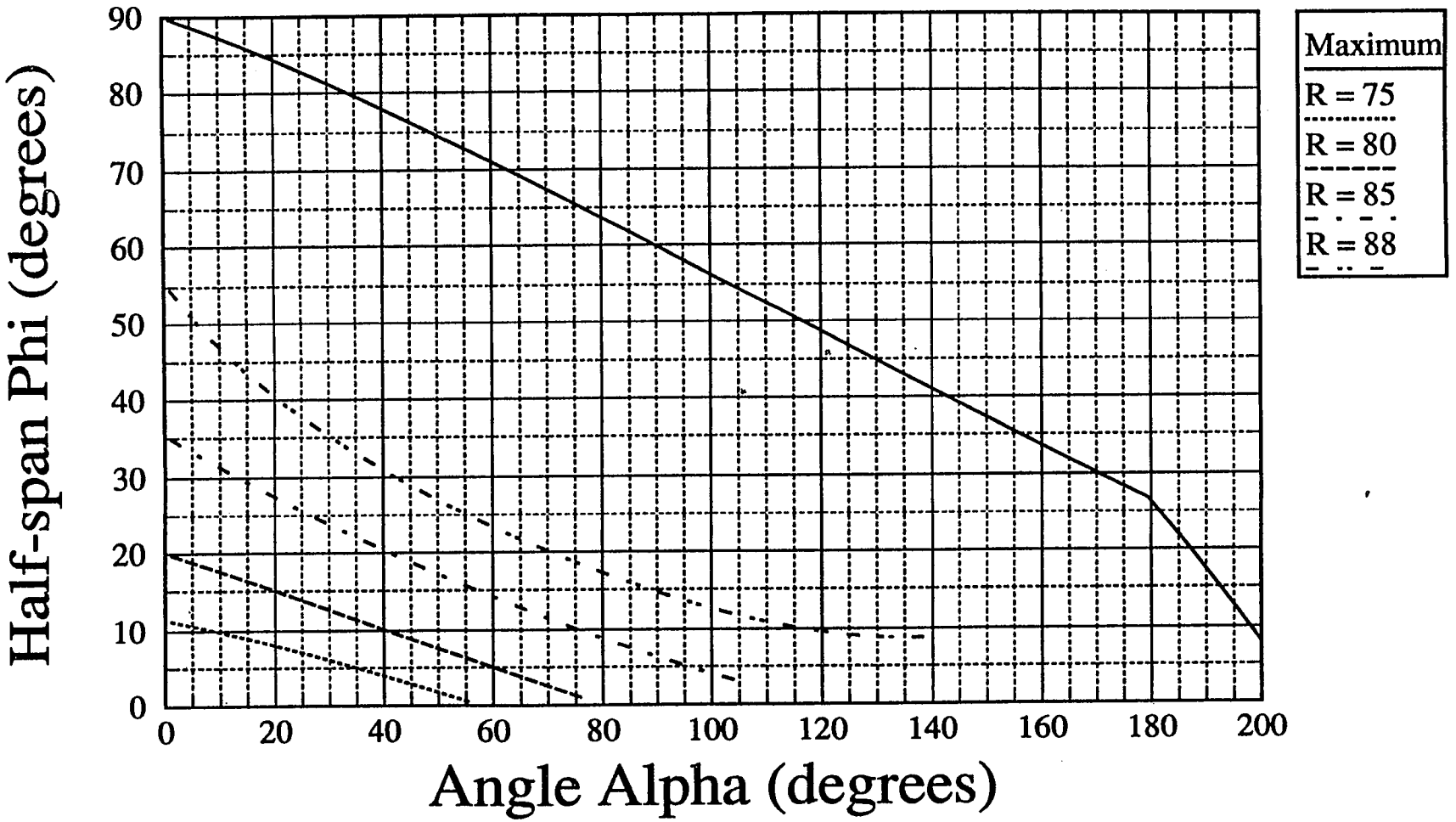


Figure 29: 1% Dwell Ranges for RRSC.1ad (B = 1.0)

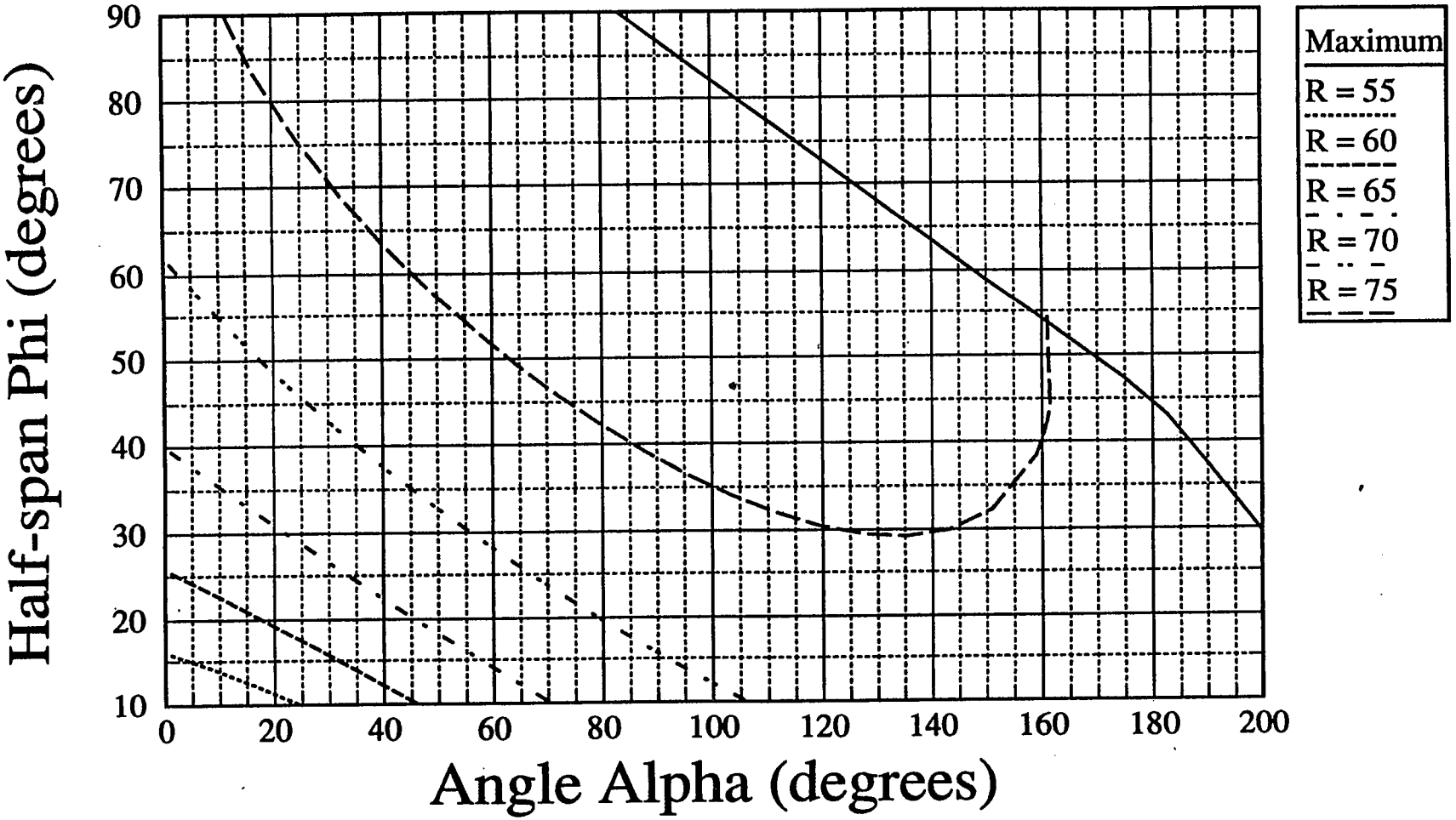


Figure 30: 5% Dwell Ranges for RRSC.1ad (B = 1.0)

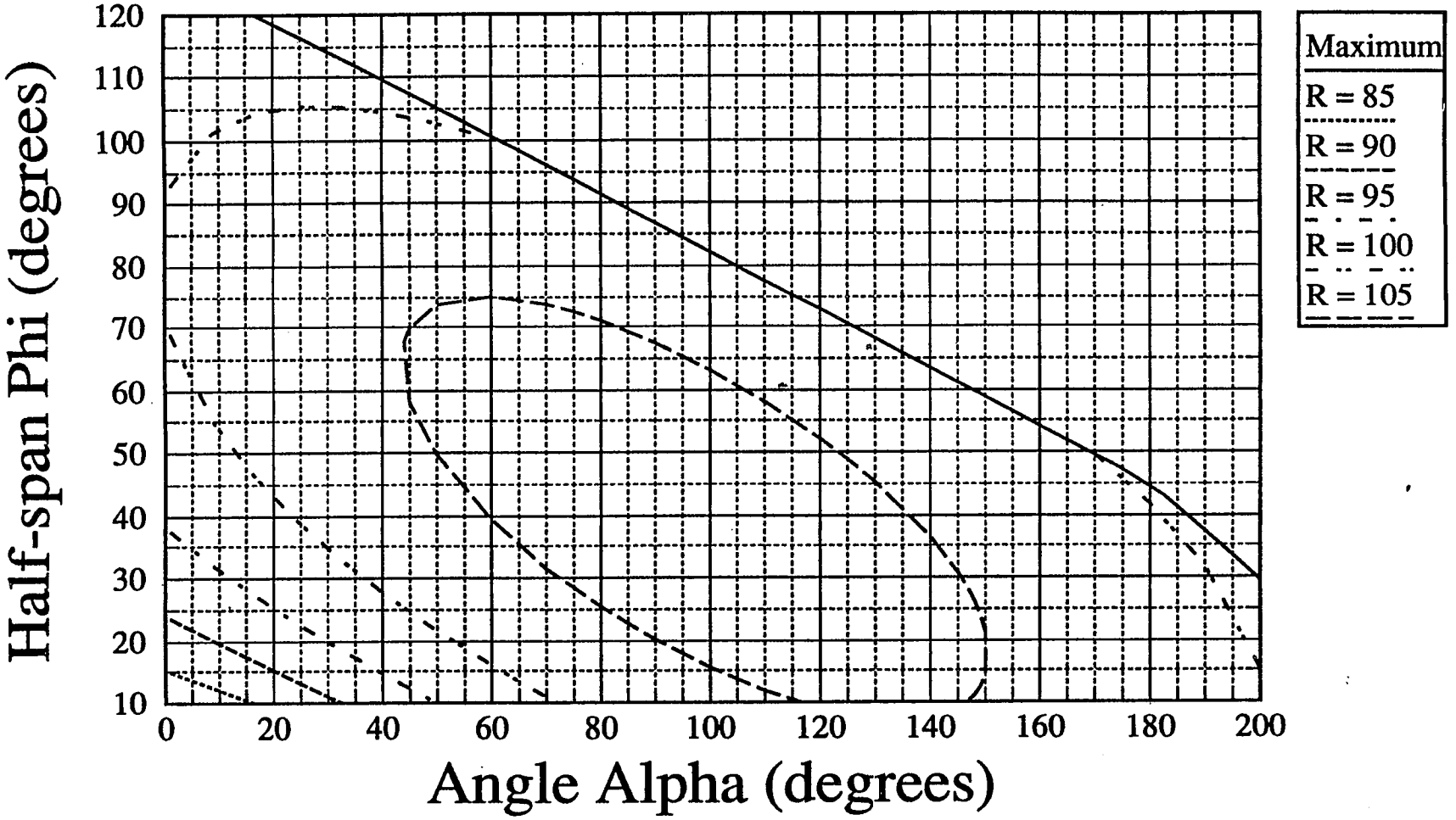


Figure 31: 1% Dwell Ranges for RRSC.1ad (B = 1.5)

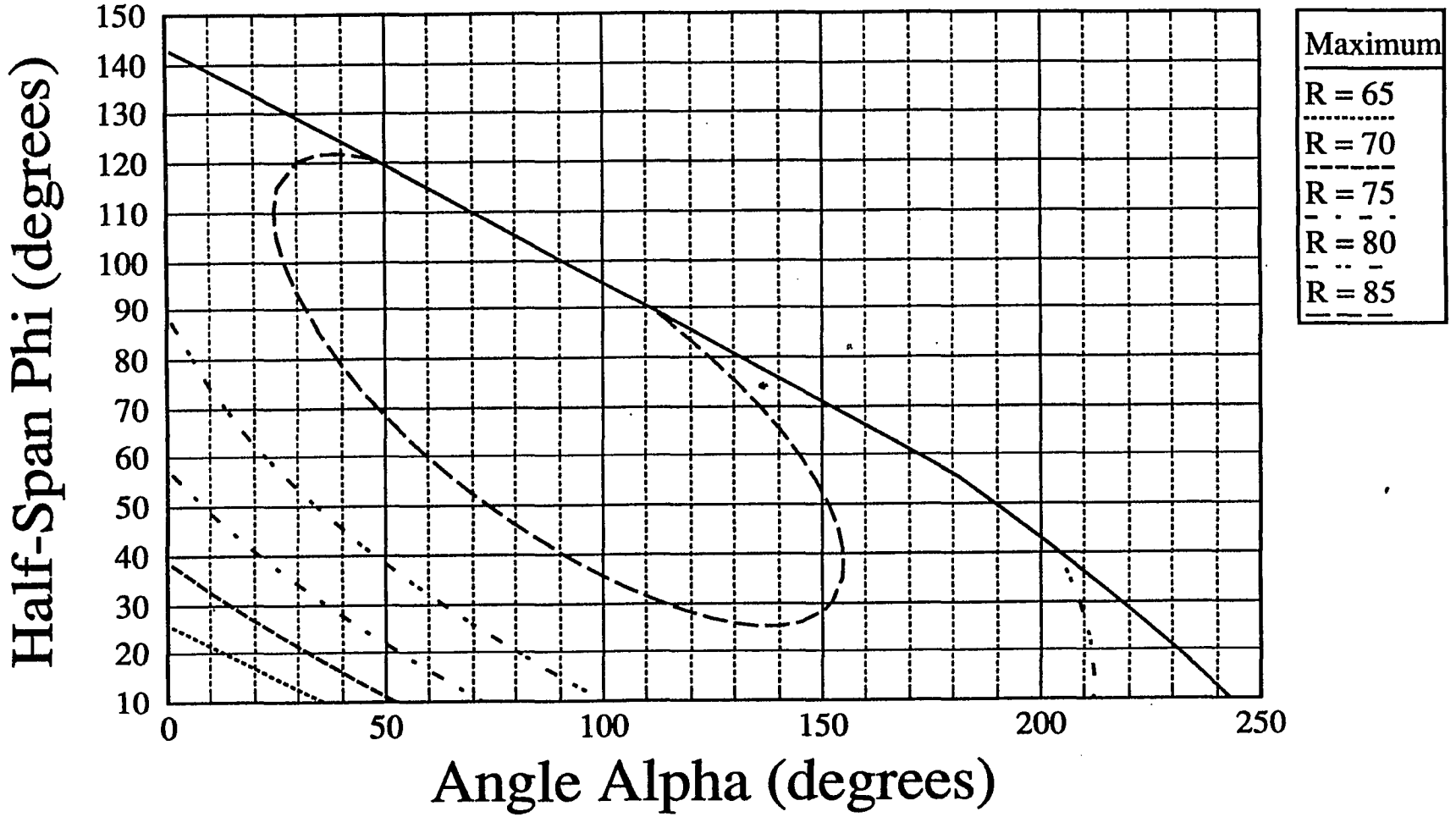


Figure 32: 5% Dwell Ranges for RRSC.1ad (B = 1.5)

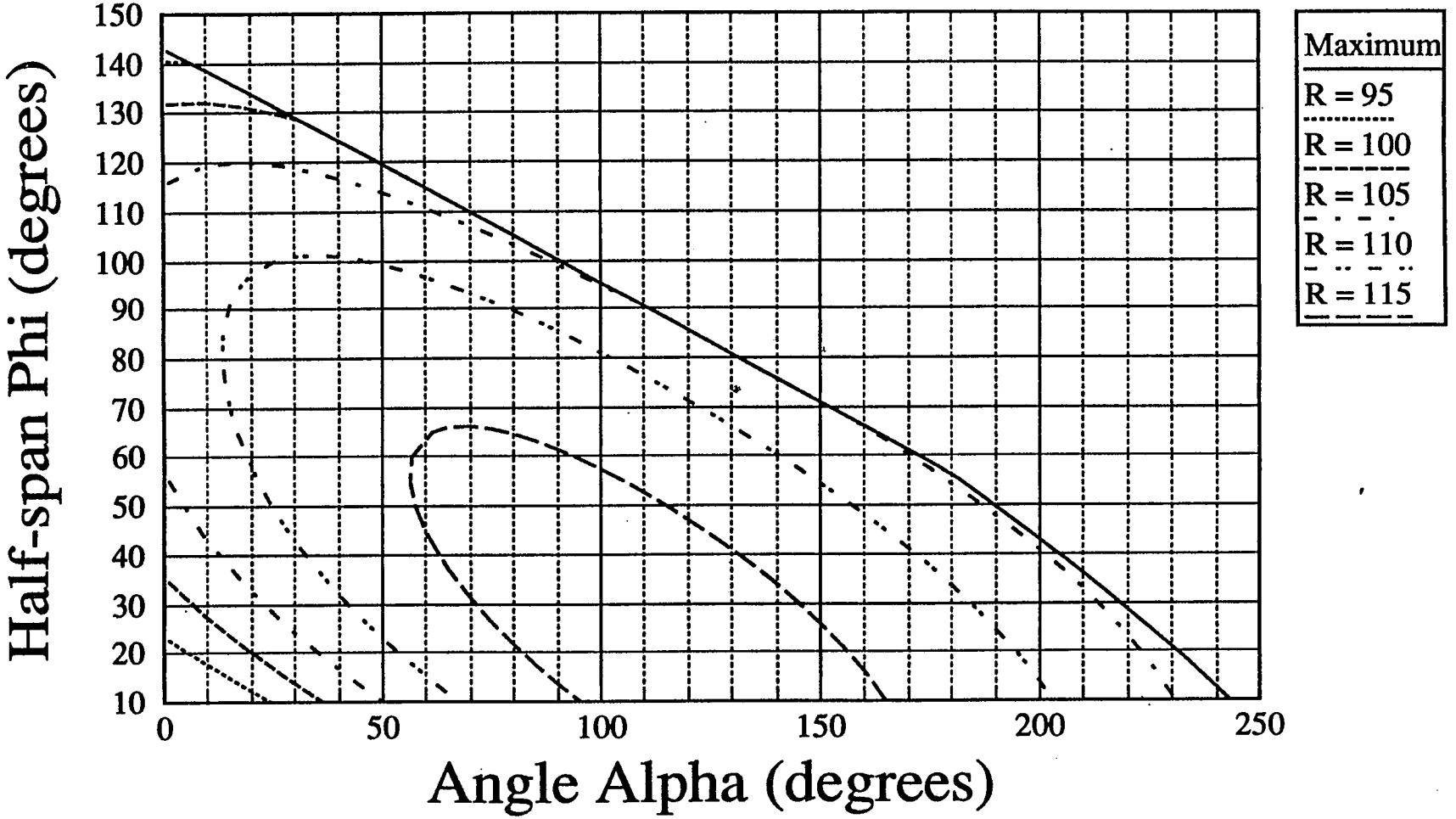


Figure 33: 1% Dwell Ranges for RRSC.1ad (B = 2.0)

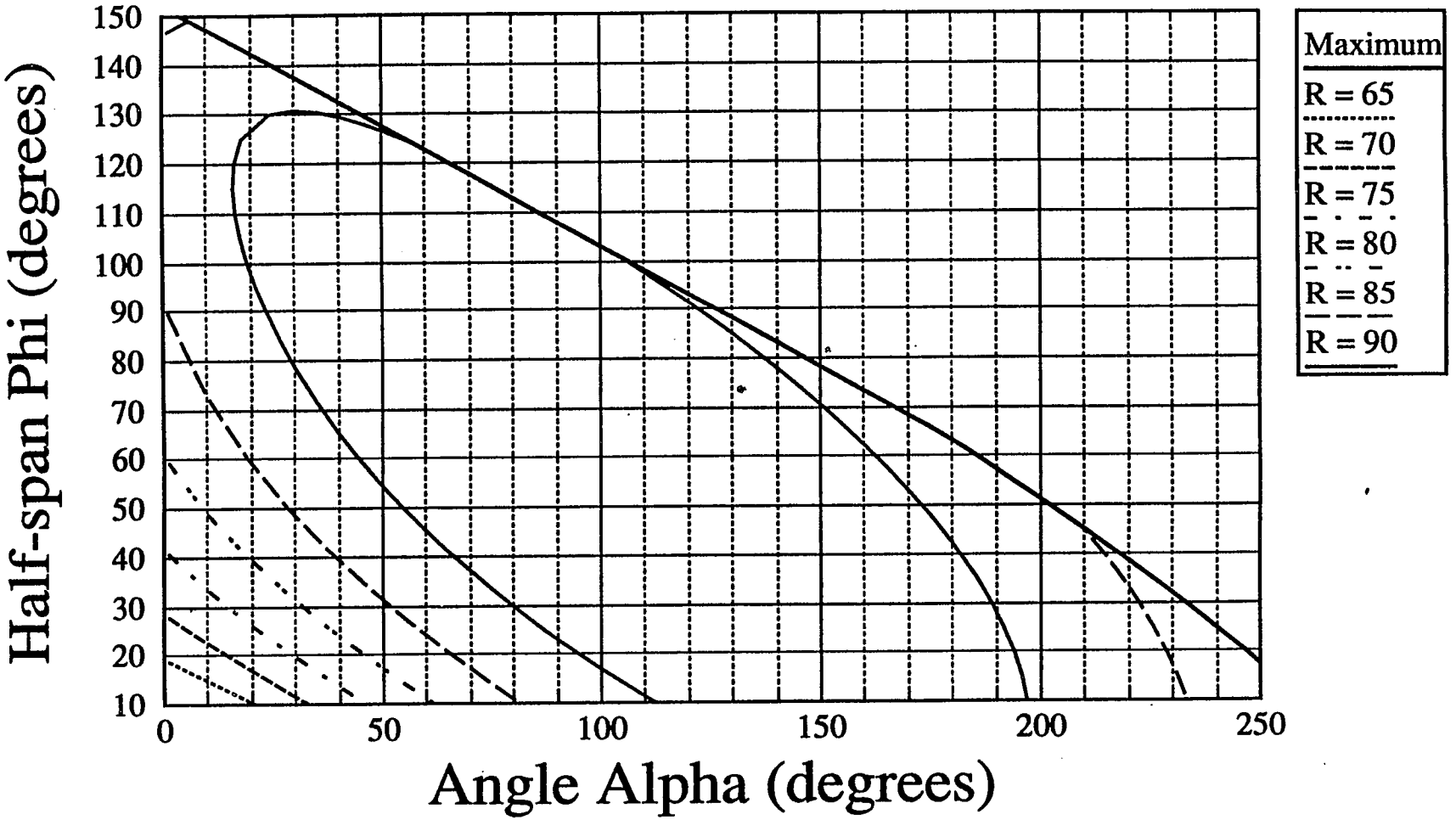
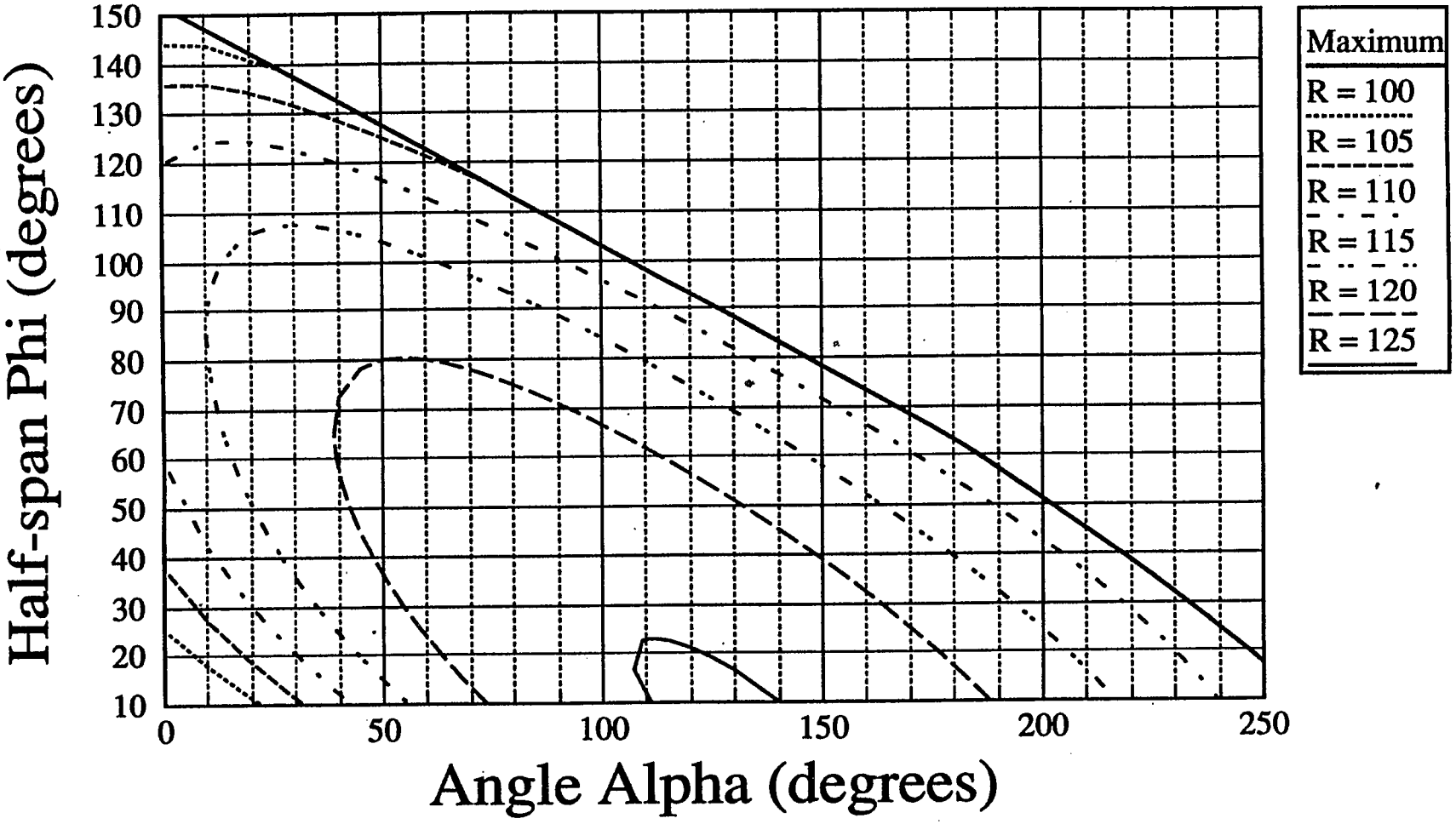


Figure 34: 5% Dwell Ranges for RRSC.1ad (B = 2.0)



3.2 The RSRC Mechanism

Another linkage configuration that will produce dwell motion is the RSRC linkage. The general configuration is given in Figure 35. The revolute R_{12} is the input joint, while the output is the translational component of C_{41}. Crank length r, Link 4 length q, vertical offset f, and cylinder displacement p are all normalized with respect to coupler length b to produce R, Q, F, and P; the design space is (R, Q, F). This linkage configuration was also identified and studied by Hunt and Shrivastava [19].

This linkage experiences a dwell in the cylinder displacement P at a crank angle of zero. If the translational component of C_{41} is held stationary, the reachable space of $C_{14}R_{43}$ is a torus; revolute R_{12} produces a circle as its reachable space. According to Hunt and Shrivastava, a section Z = F taken through the torus such that

$$Q - 1 < F < Q \tag{39}$$

will have a "dumb-bell"-shaped curve, and this curve will osculate with the circular crank path if

$$R = F\,(1/(Q - F)^2 - 1)^{0.5} \tag{40}$$

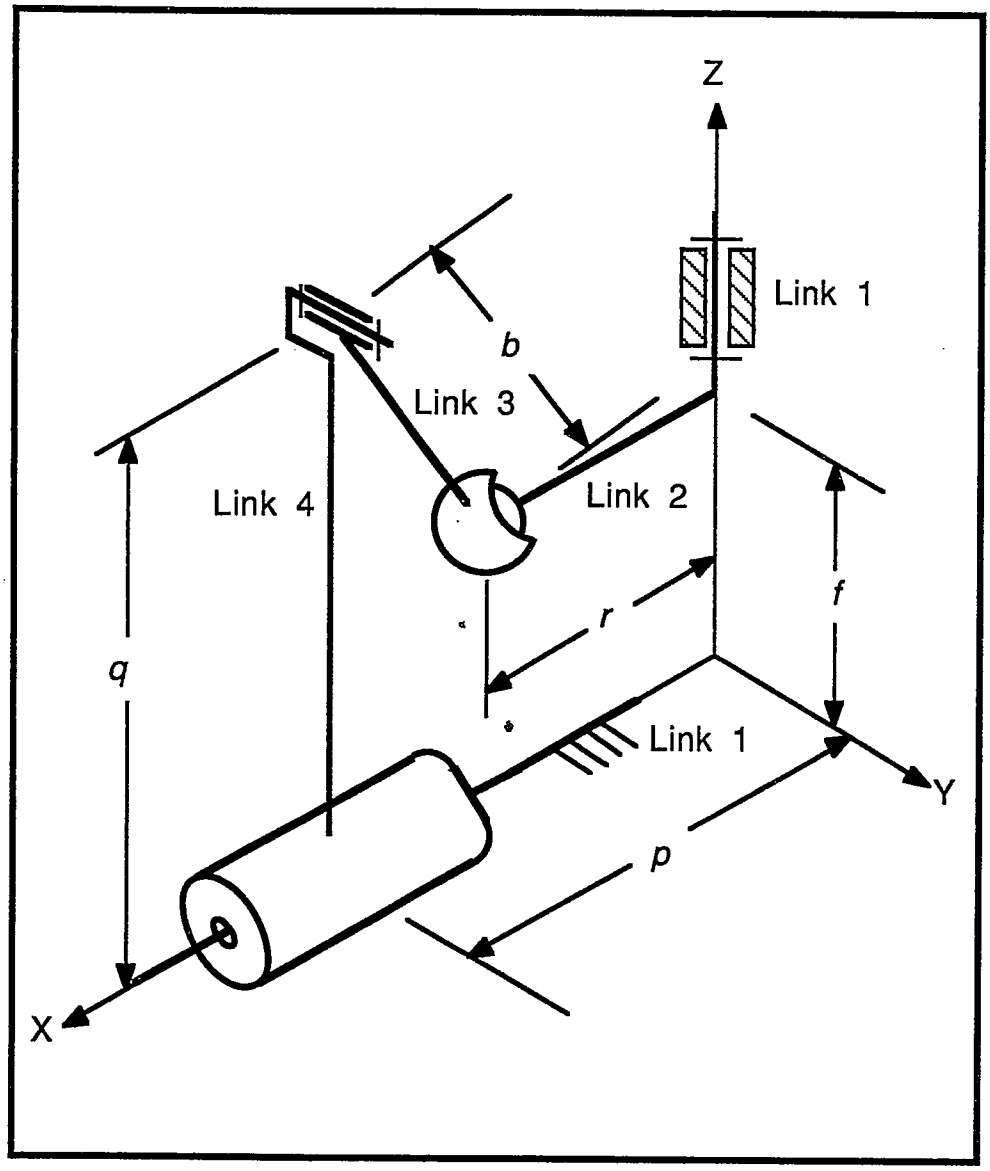


Fig 35: RSRC Configuration, with definitions of Link Lengths

The inequality constraint ensuring full crank rotation is

$$R^2 < (Q + 1)^2 - F^2 \tag{41}$$

This configuration will have two parameters, since it has a three-dimensional design space and one equality constraint. The design variable Q was chosen as one parameter, and a new variable $Z = Q - F$ was chosen as the other. The relationship

$$F = Q - Z \tag{42}$$

allows inequality (40) to be rewritten as

$$0 < Z < 1 \tag{43}$$

and equation (41) as

$$R = (Q - Z)\,(1 - Z^2)^{0.5} / Z \tag{44}$$

The relationship between P and crank angle θ is given in [19], but this relationship is incorrect. The correct relationship is derived here.

The position of S_{23}, in XYZ coordinates, is ($R \cos \theta$, $R \sin \theta$, F). The length of

Link 3 forces the position of R_{34} to satisfy

$$(P - R\cos\theta)^2 + (Q\cos\mu - R\sin\theta)^2 + (Q\sin\mu - F)^2 = 1 \qquad (45)$$

where μ is defined as shown in Figure 36. Since

$$\sin\mu = R\sin\theta / (F^2 + R^2\sin^2\theta)^{0.5} \qquad (46)$$

and

$$\cos\mu = F / (F^2 + R^2\sin^2\theta)^{0.5} \qquad (47)$$

Equation (45) can be rewritten as the quadratic

$$P^2 - \beta P + \gamma = 0 \qquad (48)$$

where

$$\beta = 2R\cos\theta \qquad (49)$$

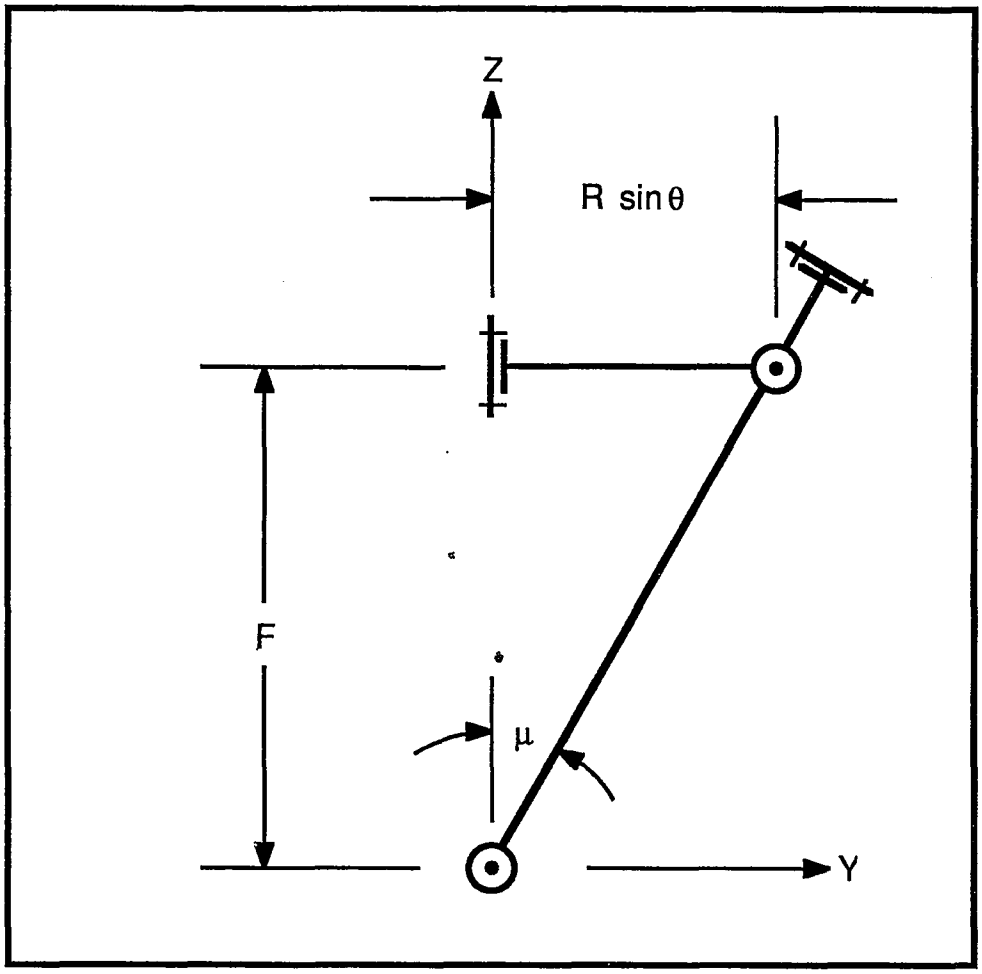


Figure 36: Projection of RSRC onto YZ Plane

and

$$\gamma = Q^2 + F^2 + R^2 - 2\,Q\,(F^2 + R^2 \sin^2\theta)^{0.5} - 1 \qquad (50)$$

Equation (39) can be solved for P with the quadratic formula:

$$P = \beta / 2 + (\beta^2 - 4\,\gamma)^{0.5} / 2 \qquad (51)$$

Hunt and Shrivastava give the value of P at the dwell point as

$$P_0 = RQ / F \qquad (52)$$

and the total span of P as

$$\text{span} = 2\,R \qquad (53)$$

Equations (42), (44) and (49) - (53) were combined in a TK Solver file to produce

a model of this linkage. Dwell range and tolerance were computed using

$$\text{range} = 2\,\theta \tag{54}$$

$$\text{tolerance} = (P_o - P) / \text{span} \tag{55}$$

Curves of constant dwell range for tolerances of one and five percent were calculated, and are given in Figures 37 and 38.

3.2 The RSCP Mechanism

This linkage configuration was identified by Hunt and Shrivastava as having dwell potential, but a dwell linkage was not developed. One configuration that will produce dwell motion is given in Figure 39. The revolute R_{12} is the input joint, and the slider P_{41} is the output joint. Coupler length b, vertical offset p, and slider displacement d are all normalized with respect to crank length q to produce B, P, and D. The design space consists of B, P, and α, the angle that the axis of R_{12} makes with the Z-axis.

The open kinematic chains made by R_{12} and $P_{14}C_{43}$ may be used to find the conditions for a dwell output. The output will experience a dwell at its minimum displacement when the crank angle is zero; if the slider is held stationary at this point, the generated surface of the output is an infinite cylinder. The path of the crank must osculate, at crank angle 0°, with a circular section through the cylinder. The projection of the crank path onto the XY-plane, as shown in Figure 40, is an ellipse with a major

Figure 37: 1% Dwell Ranges for RSRC

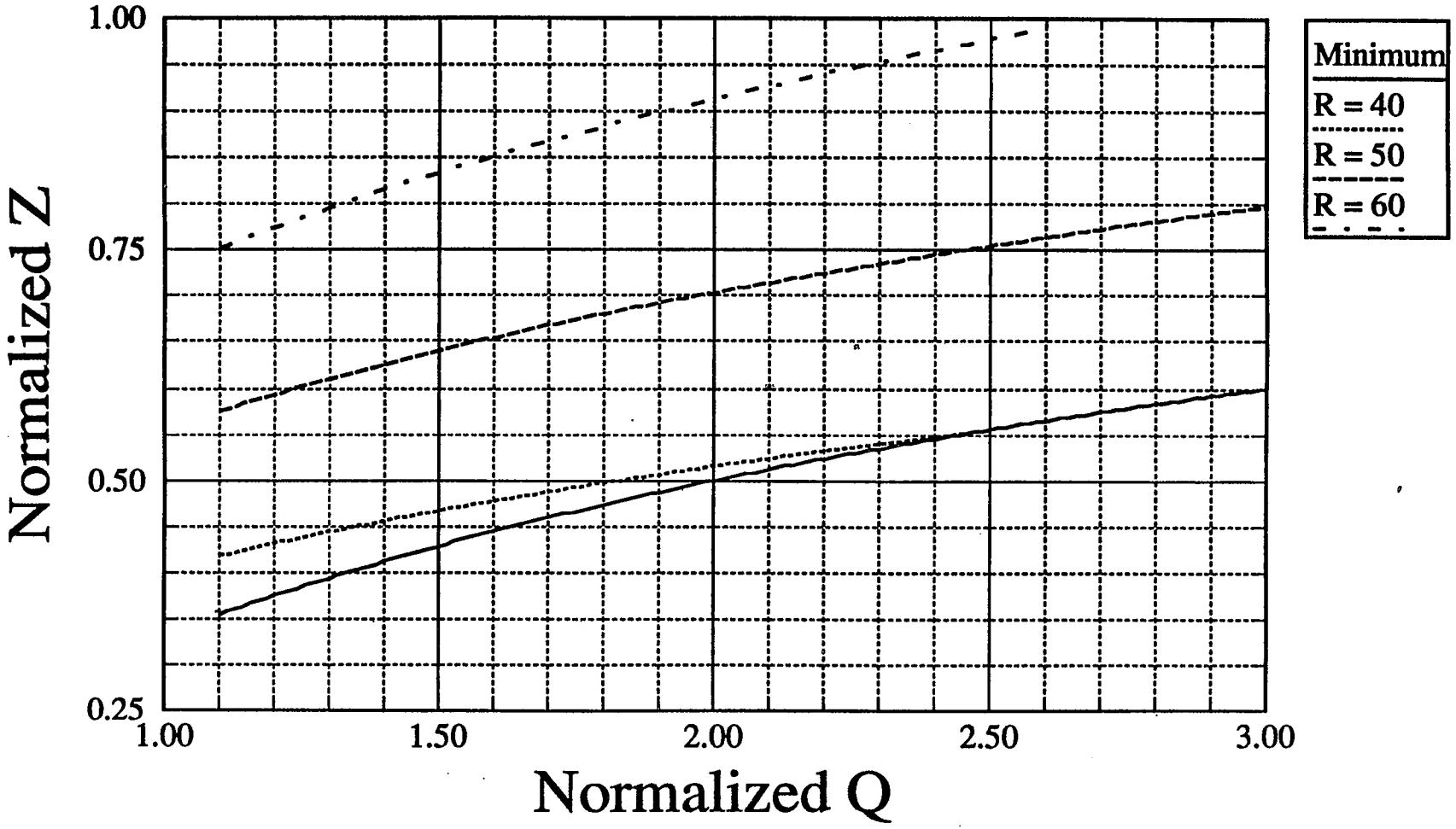
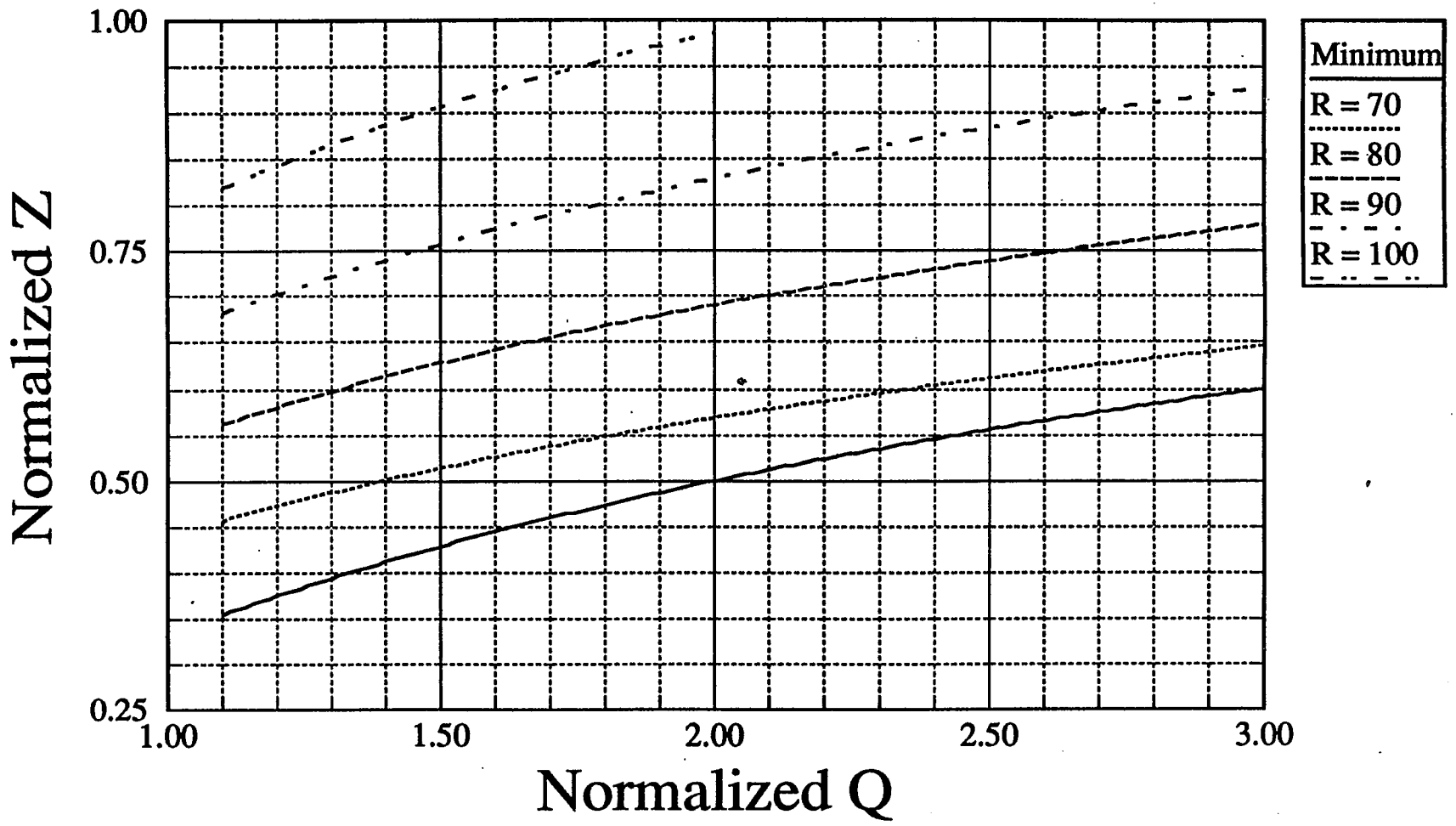


Figure 38: 5% Dwell Ranges for RSRC



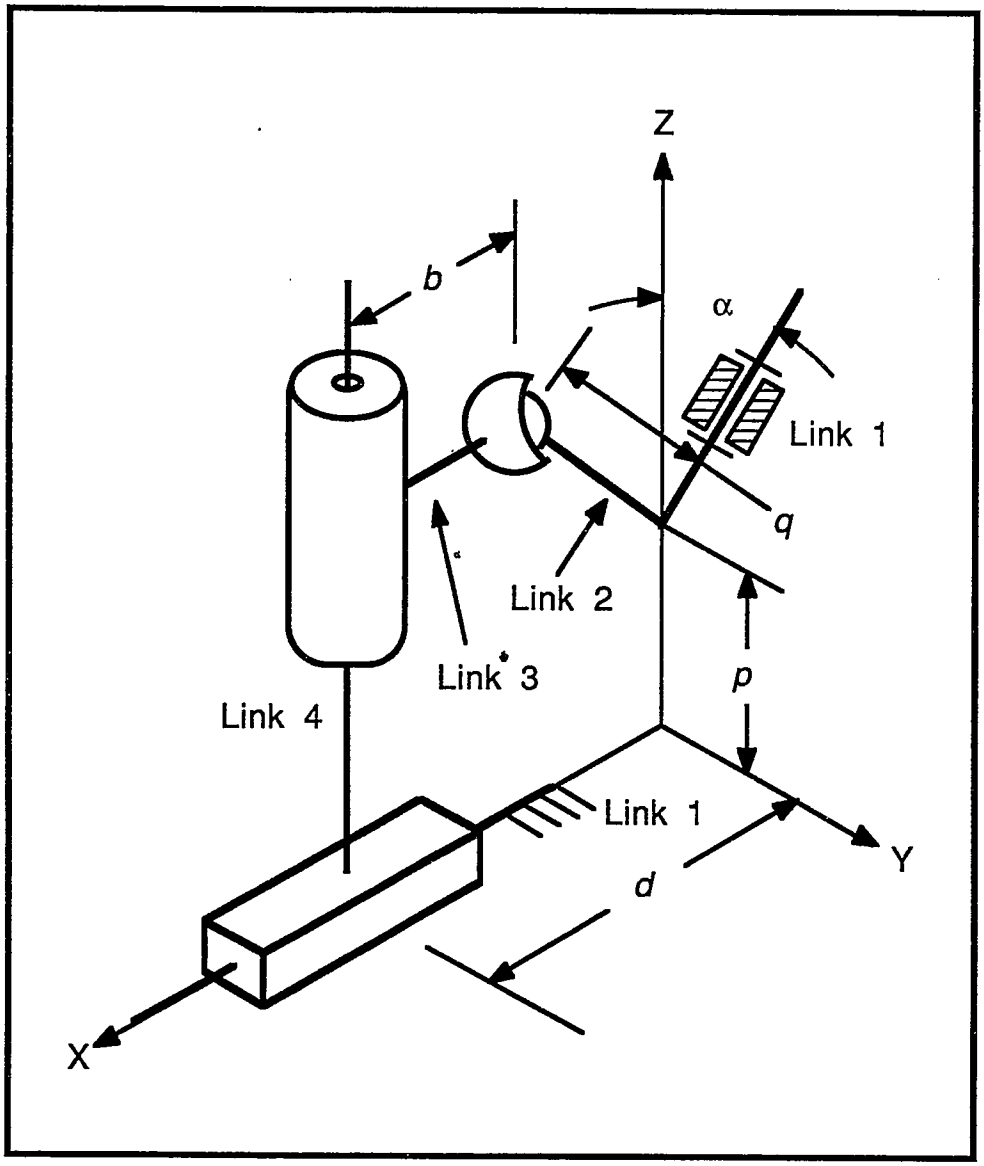


Figure 39: RSCP Configuration, with definitions of Link Lengths

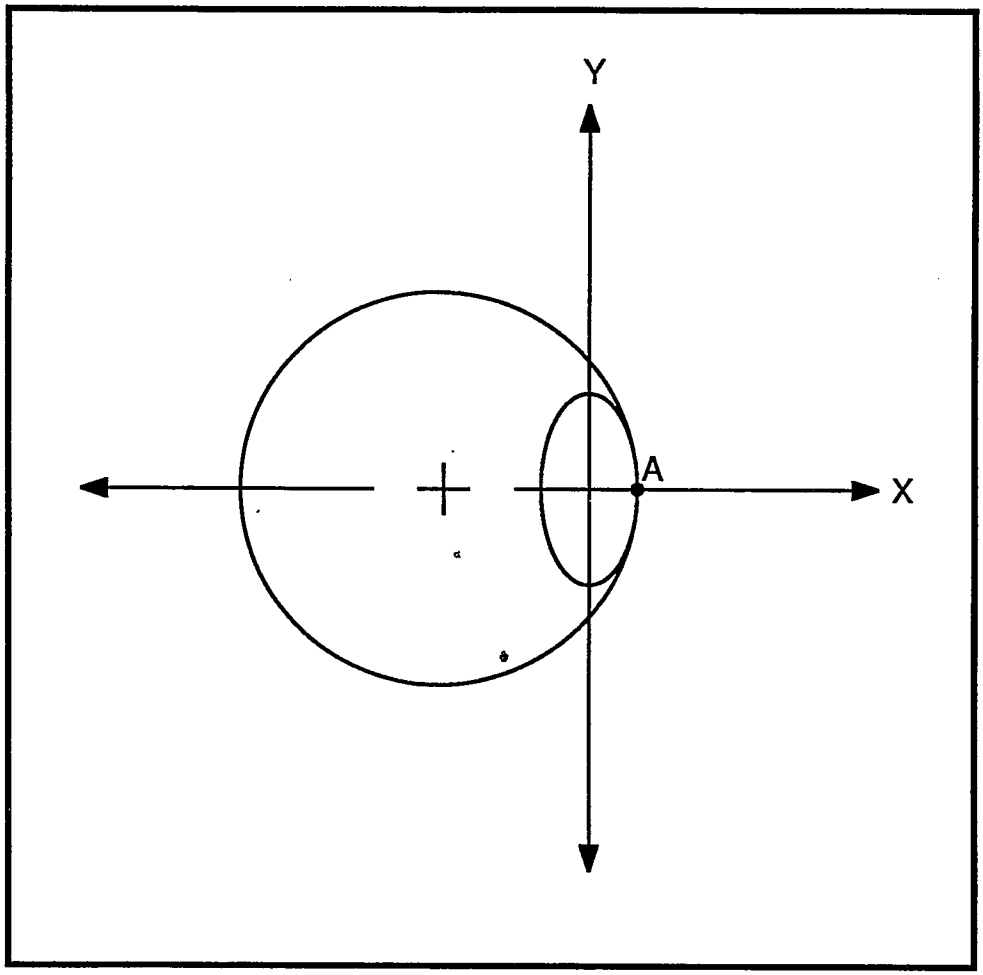


Figure 40: Elliptical Path Osculates with Circle at Point A

axis of 2 and a minor axis of 2 cos α, the equation of this ellipse is

$$X^2 / \cos^2 \alpha + Y^2 = 1 \qquad (56)$$

The crank path will osculate properly if the radius of curvature of the ellipse at $X = 0$ is equal to B, that is,

$$B = \sec \alpha \qquad (57)$$

This is the equality constraint for the RSCP dwell mechanism, and gives B as a function of α. One constraint to ensure proper linkage motion in this case is that α can never be zero, as two possible branches would then share a common position at crank angle zero, and the linkage motion would be indeterminate. For full crank rotatability, the only constraint is that B must be greater than or equal to one, and that is assured by equation (57) and a nonzero α.

There are three design variables and one equality constraint, so two design parameters are expected; α and P may be used as the design parameters. Since it will be shown below that P does not affect the dwell constraint or the functional relationship between input and output, that is, since P has no affect on the functional requirements considered here, the choice of P may be ignored in this thesis. Thus α is the only necessary design parameter, and the parameter space has only one dimension.

The input/output relationship may be found by considering the projection of the

linkage onto the XY-plane, as shown in Figure 41. The position of the spherical pair follows the path of an ellipse:

$$X = \cos\theta \cos\alpha \tag{58}$$

$$Y = \sin\theta \tag{59}$$

where θ is the crank angle. Link length B is then the hypotenuse of a right triangle, one of whose sides is Y, and so

$$D = (B^2 - Y^2)^{0.5} - X = (B^2 - \sin^2\theta)^{0.5} - \cos\theta \cos\alpha \tag{60}$$

Since Y is zero at the maximum and minimum values for D, the span is

$$\text{span} = 2 \cos\alpha \tag{61}$$

The dwell ranges and tolerances were given by

$$\text{range} = 2\theta \tag{62}$$

$$D = (1 - \text{tolerance}) \text{ span} \tag{63}$$

Equations (57) - (63) were combined in a TK Solver model and the ranges for one

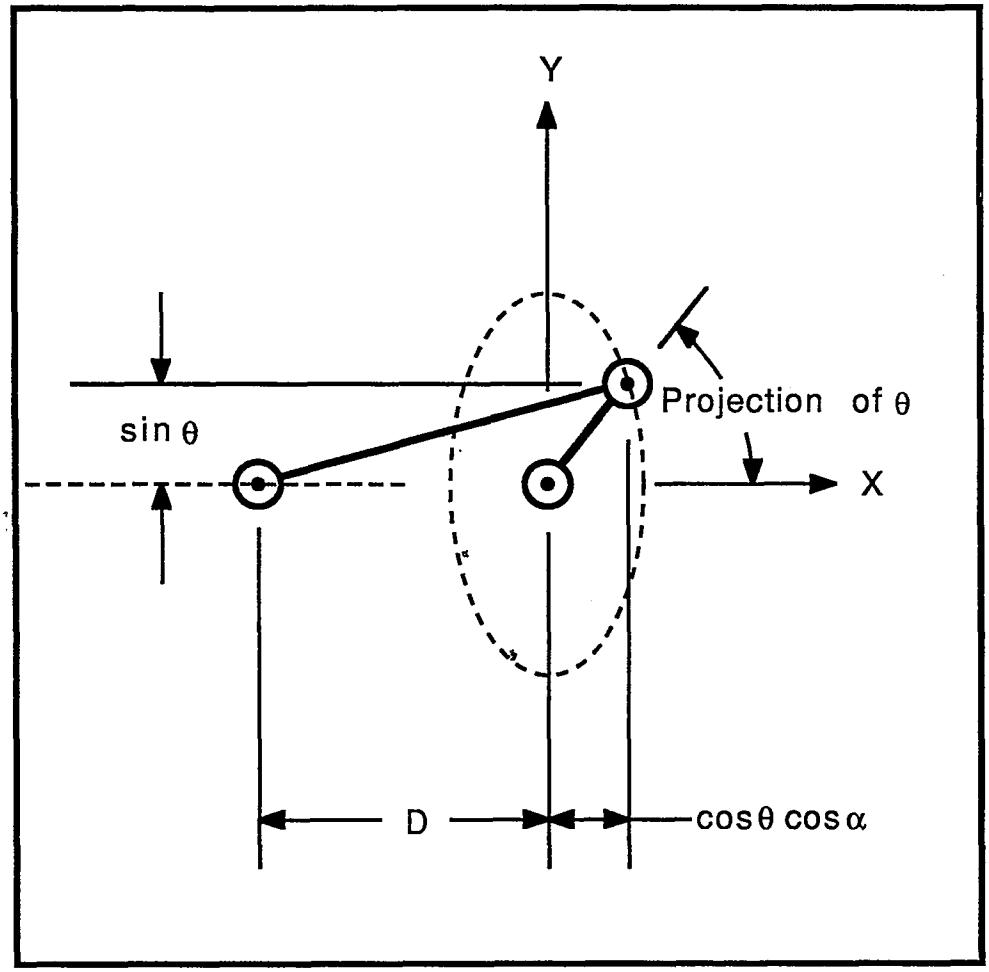
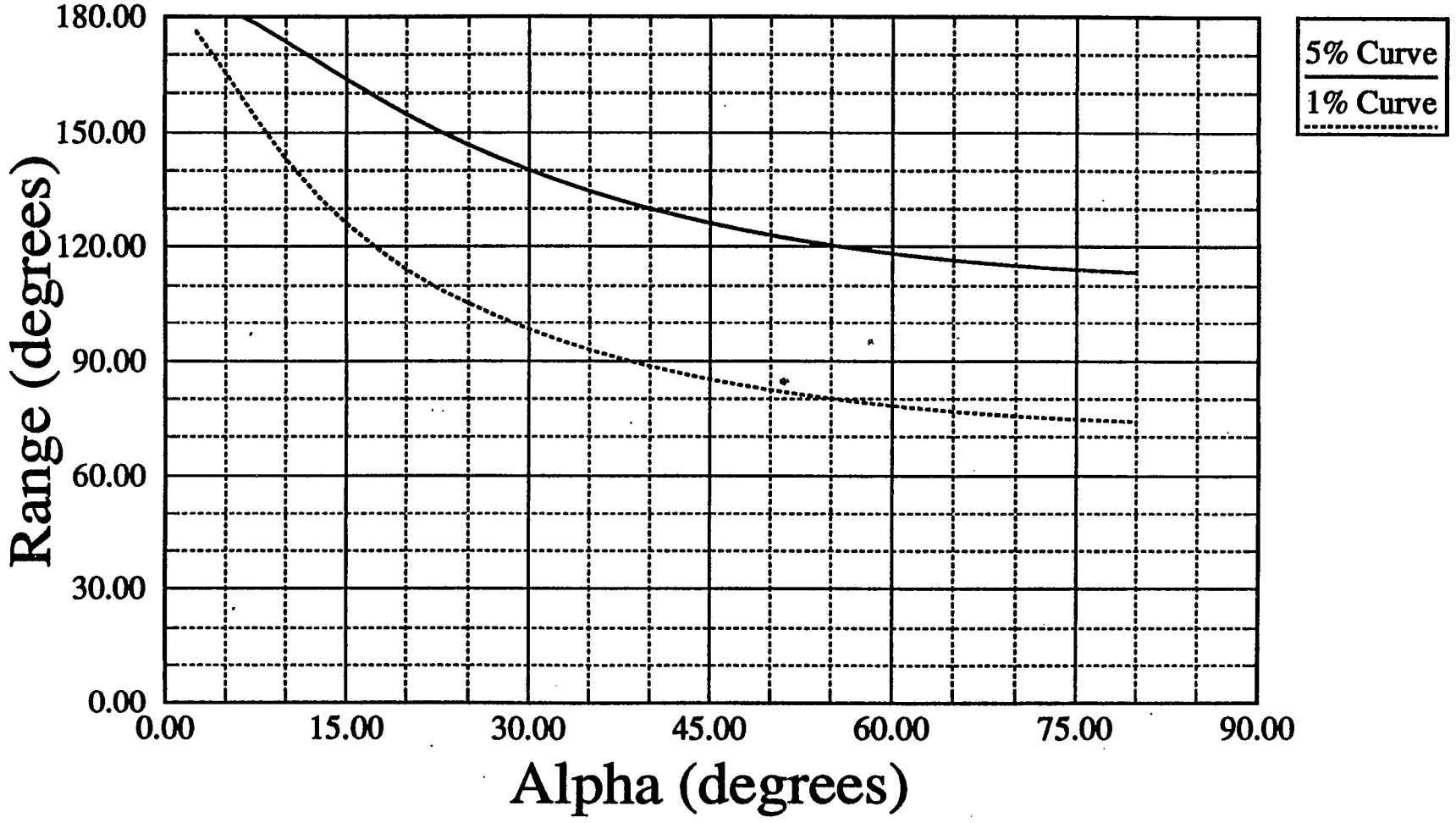


Figure 41: Projection of RSCP onto XY Plane

and five percent dwells were found for different values of α. These ranges are given as curves in Figure 42, and they may be used to design a dwell linkage.

Figure 42: Dwell Ranges for RSCP



CHAPTER FOUR

DISCUSSION OF RESULTS

The method developed in this study was applied to three mechanism configurations, and generated design procedures for six types of dwell function generator: one quadruple dwell, one double angular dwell, one double translational dwell, and one single angular dwell from the RRSC configuration, and a single translational dwell from each of the RSRC and RSCP configurations. The quadruple dwell RRSC.4d mechanism is unique among these linkages since its output is distributed across two output functions. Each output function among these linkages is symmetric about the dwell points, and the dwells on any one output function are evenly spaced and identical.

New parameters were introduced for these linkages. A single design parameter was introduced for the quadruple dwell mechanism and for the RSCP dwell mechanism, two design parameters were introduced for each of the double angular dwell, double translational dwell, and RSRC dwell mechanisms, and three were introduced for the single angular dwell mechanism.

4.1 Design for Dwell Using the Tables

These linkages may be used to design for certain dwell functions. To apply these mechanisms to the design of a dwell function generator, the designer must first decide on the nature of the output function, particularly on the number and placement of the dwell regions, and on the type of output (translational or rotational). Then the dwell

ranges and tolerances must be chosen, and the total span of the function. (For translational dwell, a choice must also be made between the two types of span available, an absolute span or a normalized span.) With these design decisions the functional requirements are completely determined.

The choices of output function type and number of dwell points determine the specific linkage configuration, thus the design parameters, to be used. Other functional requirements such as span and dwell ranges are met by the proper choice of these design parameters. Other design parameters that may be introduced are the degree of amplification (if any) applied to the output, and the overall size of the linkage. The design charts are collected in Appendix A for convenience; some examples of design are given here to illustrate their use.

4.1.1 Double Angular Dwell An angular output function is required that contains two identical dwells 180° apart, at the maximum and minimum values for the function. The span is to be 100°, and the dwell ranges are to be 80°, with an acceptable deviation of 5% of the span.

This function may be synthesized by the RRSC.2ad mechanism. The half-span phi is 50° and the dwell range is 80°; on the design chart for 5% dwell ranges for RRSC.2ad (Figure 17), this is satisfied by a B of 0.3. The parameter vector is thus

fully defined, and the link lengths may be found from Equations (7) and (20):

$Q = 1.000$

$R = \csc 50° = 1.305$

$P = \cot 50° + 0.3 = 1.139$

$B = 0.300$

These are the normalized link lengths and may be scaled to any convenient size.

4.1.2 Double Translational Dwell A translational function is required that contains two identical dwells 180° apart. The span must be four times the input crank length and the dwell range is to be 70°; a tolerance of 5% of the span is acceptable.

This function may be produced by the RRSC.2td linkage. From Figure 20, the parameter P for a normalized half-span of two and a 70° dwell range is 0.68. The normalized link lengths may be found from Equations (25) and (26):

$P = 0.68$

$R = (2)\ (0.68) = 1.36$

$B = (2 - 1)\ (0.68^2 + 1)^{0.5} = 1.21$

4.1.3 Single Angular Dwell A rotational function containing one dwell is required, with a range of 85° and a span of 120°; a tolerance of 1% is acceptable.

An RRSC.1ad linkage will be used to solve for this function. The half-span is 60°, the dwell range is 85°, and the tolerance is 1%; several solutions are possible. From Figure 31, two solutions are found ($\theta = 60°$ and $\theta = 145°$), and from Figure 33, a third solution is found at $\theta = 20°$. The three linkages are found from Equations (31) - (33) and (36):

Solution 1: $B = 1.5$, $\theta = 60°$, $\psi = 60°$ ($\tau = 0.5774$):

$B = 1.5$

$H = 1.5 (\sin 60° + 0.577 + 0.577 \cos 60°) / 2 = 0.866$

$P = 1.5 (1 + \cos 60° + 0.577 \sin 60°) / 2 + 0.577 = 2.08$

$R = [(1 + 0.866)^2 + (2.08 - 1.5)^2]^{0.5} = 1.95$

Solution 2: $B = 1.5$, $\theta = 145°$, $\psi = 60°$ ($\tau = 0.5774$):

$B = 1.5$

$H = 1.5 (\sin 145° + 0.577 + 0.577 \cos 145°) / 2 = 1.22$

$P = 1.5 (1 + \cos 145° + 0.577 \sin 145°) / 2 + 0.577 = 0.961$

$R = [(1 + 1.22)^2 + (0.961 - 1.5)^2]^{0.5} = 2.28$

Solution 3: $B = 2.0$, $\theta = 20°$, $\psi = 60°$ ($\tau = 0.5774$):

$B = 2.0$

$H = 2.0 (\sin 20° + 0.577 + 0.577 \cos 20°) / 2 = 0.377$

$P = 2.0 (1 + \cos 20° + 0.577 \sin 20°) / 2 + 0.577 = 2.71$

$R = [(1 + 0.377)^2 + (2.71 - 2.0)^2]^{0.5} = 1.55$

4.1.5 Single Translational Dwell (Absolute Span) A translational function is required that contains a dwell at one extreme position. The span needed is three inches (an absolute span), and the dwell range is 100°, with a tolerance of 1% required.

This may be solved by the RSCP linkage. The design chart in this case is Figure 42, where a 120° dwell range with 1% tolerance is satisfied by an α of 17.5°. The normalized span is therefore 1.907, and so the linkage must be scaled up by choosing the crank length to be 1.573". The link length p may arbitrarily chosen as 1" to produce the linkage:

$\alpha = 17.5°$

$r = 1.573"$

$p = 1"$

$b = r \sec 17.5° = 1.649"$

4.1.6 Single Translational Dwell (Normalized Span) A translational function is required that experiences a single dwell. The normalized span is to be two, and the dwell range is to be 50°, with an acceptable tolerance of 1%.

This function may be produced by the RSRC linkage. From Figure 37, a 50° dwell range will be produced by the linkage when Q = 2.0 and Z = 0.70. The link lengths are found from Equations (42) and (44):

$$Q = 2.0$$

$$F = 2.0 - 0.70 = 0.30$$

$$R = (2.0 - 0.70)\,(1.0 - 0.70^2)^{0.5} / 0.70 = 0.306$$

$$\text{span} = 0.612$$

This linkage does not yet produce the desired output; the function must be cascaded through an amplifting mechanism. The amplification factor should be 2.0/0.612 = 3.27.

4.2 Results in Terms of Axiomatic Design

A convenient way to categorize the dwell linkages studied here is through the conceptual framework of Axiomatic Design, particularly the Independence Axiom. The three categories a design may fall into, in terms of the Independence Axiom, are uncoupled designs, where the design parameters are independent of each other, coupled designs, where the design parameters are not independent of each other, and decoupled or quasicoupled designs, where the design parameters may be considered independent

when solved for in the proper order. Uncoupled designs are the most desirable, and coupled designs are the least desirable.

The choice of which linkage configuration to use determines the subsequent number and types of choices; this choice may thus be considered to be at a more antecedent level in the design heirarchy. Once a linkage configuration is chosen, one design parameter (the configuration) and one functional requirement (the overall shape of the output function) may be eliminated from consideration; the remaining functional requirements are the span of the output function and the dwell ranges. (There is normally one dwell range, thus one functional requirement, for each dwell, but the output functions in this study have only one dwell range each since all the dwells on a single output function are identical.) The remaining design parameters are those introduced in the analysis of the linkage and an amplification factor on the output function, if the output is cascaded through an amplifying linkage. The total scale of the linkage, that is, the size of the normalizing link, may also be considered a design parameter, especially for the translational dwell mechanisms with an absolute output span, but is usually left to subsequent levels of the design heirarchy. The design parameters and functional requirements of the dwell linkages may be examined to determine the type of design.

The quadruple dwell mechanism RRSC.4d has, other than the scaling factor, three design parameters: the half span phi and two output amplification factors. It has four functional requirements: span and dwell range for each of its output functions. Thus this linkage configuration cannot be made independent since there are more functional requirements than design parameters; the same design parameter determines both dwell ranges, and the design is coupled.

The double angular dwell linkage RRSC.2ad has two design parameters; the two functional requirements are the span and the dwell range. Since the span is solely a function of the half span parameter phi, and the dwell range is a function of both phi and the other parameter B, this design is decoupled. By the same logic, the double translational dwell linkage RRSC.2td is also decoupled; a scaling factor may be added for absolute spans, making the design less than optimum since there are more design parameters than functional requirements.

The RRSC.1ad single angular dwell linkage has three design parameters, one of which is a half span parameter. Since the only functional requirements are the span and the dwell range, and the span is determined solely by the half span parameter while the dwell range is determined by all three parameters, this design is also decoupled. However, this design also is less than optimum since there are more design parameters than need be.

The RSRC single translational dwell linkage has two functional requirements and two design parameters introduced in the analysis; since both span and dwell range are functions of both design parameters, this is a coupled design. The design can be decoupled by introducing a scaling factor for absolute spans or an amplification factor for normalized spans; however, the resulting decoupled design now has more design parameters than functional requirements and so is less than optimum.

The RSCP single translational dwell linkage has two functional requirements; the three design parameters for this linkage are the parameter introduced in the analysis, a scaling or amplification factor for the span, and the normalized link length P, which has no affect on the functional requirements. This is also a decoupled design with one extraneous design parameter.

The linkages studied here, with the exception of the quadruple dwell linkage, are all decoupled. In certain regions of the parameter space, the dwell curves for several of the linkages approximate horizontal lines, and in those regions the designs are approximately uncoupled. This is the case for RRSC.2ad when ψ is between 45° and 120° and 1% dwell range is between 60° and 80°, and when ψ is between 45° and 120° and 5% dwell range is between 80° and 100°; and for RRSC.2td when T_{max} is between 1.5 and 2.5 and the 5% dwell range is between 40° and 70°. The RRSC.1ad design curves have points where the tangents are approximately hoizontal, but these regions are very restricted compared with the regions in the RRSC.2ad and RRSC.2td cases.

4.3 Other Issues

Using the design charts developed here, a limited number of dwell function generators may be synthesized. The technique developed in this study may be applied to many other linkages, however, and should be, to expand the range of dwell function designs available. Other areas that must be addressed are such requirements of good linkage design as transmission effectiveness, and issues related to dynamic effects, such as vibration and distortion of the output function. In subsequent research, these design issues may possibly be incorporated into this design paradigm as constraints that limit the acceptable regions of design space; in the meantime, the linkages synthesized using this technique may be used as a starting point for mechanism and machine design, and further analysis and refinement of the design may be required to produce a completely acceptable mechanism.

Another area that may be addressed in future research is the use of stronger and

weaker dwell requirements for the intersecting paths. The number of infinitesimally separated points shared by both curves is three in this study, thus the order of tangency is two (or, the order of osculation is one). A greater or smaller order of tangency between the curves will produce output functions with larger or smaller dwell ranges, respectively. Other dwell types that may be added are those that resume travel in the same direction after the dwell point as before it.

Several remarks may be made in closing about the methods of analysis. One is that a problem exists with the technique of reducing the design space, and that is that the equality constraints used to develop the design parameters may not describe rational curves through the design space, and so a single set of design parameters may not describe the entire locus of dwell linkages. This problem may be avoided in some cases if the acceptable region of the design space contains only one branch of the curve, but some linkage configurations will have more than one branch in the acceptable region, and will need more than one parameter vector and mapping function. In other linkage configurations, such as the RSRC configuration, a convenient parameter vector may produce non-unique points in the range for some points in the domain (that is, the mapping relationship is not a function). For the RSRC configuration, this means that, although the span is equal to twice the normalized link length R, the link length R was not used as a parameter since it would have resulted in multuple solutions for either F or Q, and the parameters were chosen otherwise.

The final remark concerns the kinematic analysis methods used in this research. The matrix technique, once a program was developed to implement it, was the more general of the two methods and could model any closed loop linkage, but when tested, the program was too slow for practical use. When the RRSC.4d linkage was analyzed,

each new position needed between three and five iterations to converge, and each iteration took about six seconds. Thus finding the joint variables for crank angles up to sixty degrees by one degree increments could take up to twenty minutes. For the mutiple analyses needed to find the design curves, this program was prohibitively slow. The problem may have been that the SpaceLink program is on a slow platform (the Macintosh Plus CPU is a Motorola MC 68000 chip with a clock rate of 7.83 megahertz) or that the nested-function way SpaceLinks implements the matrix technique slows performance. An analysis of the RRSC.4d linkage using IMP on the Mechanical Engineering Sun system ran in about ninety seconds, so the technique can be potentially fast enough for interactive mechanism design and analysis.

CHAPTER FIVE

CONCLUSION

A method was developed in this study to simplify the design of spatial dwell linkages by reducing the number of design choices; this method was applied to several spatial linkages, and produced reduced sets of design parameters for these linkages. Other techniques were developed for position analysis of spatial linkages on the personal computer, and these were applied to the linkages studied, to determine the relationships between the design parameters and the functional requirements for dwell motion design. This application resulted in sets of design charts which may be used by practicing engineers to develop dwell linkages.

This technique of reducing the design parameters appears to be a useful way of dealing with the design of dwell linkages. The resulting design charts are easily applied to design problems, and the method of identifying and analyzing linkage configurations with dwell potential, upon which the parameter reduction technique is based, may be applied to many spatial linkages. Thus design for dwell function generation may eventually be performed using a wide range of function types and linkage configurations, forming a library of easily designed mechanisms. Further, the linkages themselves may be more easily classified and evaluated according to the categories of axiomatic design.

Two methods of position analysis were applied to the linkages studied in this study. One, the matrix method, was generally applicable to any single loop linkage, but was also prohibitively slow. The other, using projective geometry, had to be

reapplied and derived separately for each linkage configuration, but resulted in simpler models that could be solved more quickly and easily. Thus the matrix program SpaceLinks would be considered only a backup for those linkages that might not be amenable to the simpler geometric approach.

More work is needed to develop this design parameter technique fully as a method for dwell function generator synthesis, but in this study the technique has been demonstrated to be valid.

CHAPTER SIX

REFERENCES

[1] Alizade, R.I., A.V. Rao, and G.N. Sandor,"Optimum Synthesis of Two-Degree-of-Freedom Planar and Spatial Function Generating Mechanisms Using the Penalty Function Approach," Polytechnic Institute of Baku, USSR, *Journal of Engineering for Industry (Trans. ASME)*, vol. 97, series B, no. 2, May 1985, pp. 629-634.

[2] Burden, R.L. and J.D. Faires, *Numerical Analysis* (3rd ed.), (Boston: Prindle, Weber, and Schmidt, 1985).

[3] Denavit, J. and R.S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics* vol. 22, June 1955, (*Trans. ASME* v. 77), pp. 215-221.

[4] Dimentberg, F.M., "A General Method for the Investigation of Finite Displacements of Spatial Mechanisms and Certain Cases of Passive Constraints," *Trudi Semin. po Teor Mash. Mekh. Akad.*, Nauk USSR, vol. 5, no. 17, 1948, pp. 5-39.

[5] Hain, K. "Design Curves for Linkages and their Use," *Proceedings of the Institute of Mechanical Engineers*, (London), vol. 186 paper no. 59 for Meet Sept. 5, 1972, pp 845-854.

[6] Harding, B.L. "Hesitation," *Journal of Engineering for Industry (Trans ASME)*, pp. 205-212, May 1965.

[7] Hunt, K.H. *Kinematic Geometry of Mechanisms*, (Oxford: Oxford University Press, 1978).

[8] Jenkins, E.M., F.R.E Crossley, and K.H. Hunt, "Gross Motion Attributes of Certain Spatial Mechanisms," *Journal of Engineering for Industry (Trans ASME)*, pp. 83-90, Feb 1969.

[9] JML Research, Incorporated, *The Integrated Mechanisms Program: Language Specification and User's Manual*, (Madison, Wisconsin: JML Research, Incorporated, 1986).

[10] Jwo, Chin-Hung and G.K. Matthew, "Five Multiply Separated Positions Dwell Linkage," University of Florida, *Design Engineering Technical Conference*, Columbus, OH Oct. 5-8, 1986. (ASME sponsored)

[11] Kislitsen, C.G., "Tensor Methods in the Theory of Spatial Mechanisms," *Trudi Semin. po Teor Mash. Mekh. Akad.*, Nauk USSR, vol. 14, no. 54, 1954, pp. 51-75.

[12] Kota, S., A.G. Erdman, and D.R. Riley, "Development of a Knowledge Base for Designing Linkage-Type Dwell Mechanisms: Part 1: Theory," *Journal of Mechanisms, Transmissions, and Automation in Design (Trans ASME)*, vol. 109, Sept 1987, pp. 308-315.

[13] Kota, S., A.G. Erdman, and D.R. Riley, "Development of a Knowledge Base for Designing Linkage-Type Dwell Mechanisms: Part 2: Application," *Journal of Mechanisms, Transmissions, and Automation in Design (Trans ASME)*, vol. 109, Sept 1987, pp. 316-321.

[14] Kota, S., A.G. Erdman, and D.R. Riley, "Minn-Dwell – Computer-Aided Design and Analysis of Linkage-Type Dwell Mechanisms," Proceedings of the International Computers in Engineering Conference and Exhibit 1987 (vol. 2) (New York: The American Society of Mechanical Engineers, 1987).

[15] Rastegar, J. "On the Derivation of Grashof-Type Movability Conditions With Transmission Angle Limitations for Spatial Mechanisms," *Journal of Mechanisms, Transmissions, and Automation in Design (Trans. ASME)*, vol. 111, pp. 519-523, Dec.1989.

[16] Sandor, G.N. and A.G. Erdman, *Advanced Mechanism Design: Analysis and Synthesis*, vol.2 (Englewood Cliffs, NJ: Prentice-Hall, 1984).

[17] Shoup, T.E., *Numerical Methods for the Personal Computer*, (Englewood Cliffs, NJ: Prentice-Hall, 1983).

[18] Shrivastava, A.K. and K.H. Hunt, "Quadruple Dwell from a Four-Bar Spatial Linkage," *Journal of Mechanisms*, vol. 6, pp. 241-245 (Great Britain: Pergamon Press, 1971).

[19] Shrivastava, A.K. and K.H. Hunt, "Dwell Motion from Spatial Linkages," *Journal of Engineering for Industry (Trans ASME)*, pp. 511-518, May 1973.

[20] Sodhi, R.S., A.J. Wilhelm, and T.E. Shoup, "Design of a Four-Revolute Spherical Function Generator with Transmission Effectiveness by Curve Matching," *Mechanism and Machine Theory* vol. 20, no. 6, pp. 577-585, 1985. (Pergamon Press Ltd.)

[21] Structural Dynamics Research Corporation, *I-DEAS: Geomod Solid Modeling and Design: User Guide*, (Milford, Ohio: Structural Dynamics Research Corporation, 1986).

[22] Suh, C.H., "Differential Displacement Matrices and the Generation of Screw Axis Surfaces in Kinematics," *Transactions of the ASME, Journal of Engineering for Industry*, Feb. 1971, pp. 1-10.

[23] Suh, N.P., *The Principles of Design*, (not yet published).

[24] Suh, N.P., et al., *Manufacturing Engineering*, (not yet published).

[25] Sutherland, G.H. and B. Roth, "Improved Least-Square Method for Designing Function-Generating Mechanisms," Ohio State University, Columbus, ASME Paper no. 74-DET-4 for Meeting Oct. 6-10, 1974.

[26] Tesar, D. and A. Chaudhari, "Cycloid Dwell Linkage Design Charts based on Four Multiply Separated Positions," University of Florida, Gainesville, ASME paper no. 74-DET-61 for Meeting Oct. 5-9, 1974.

[27] Wilhelm, A. J., *Design of Spatial Linkages for Function Generation by Curve Matching*, Master's Thesis, Wichita State University, 1984.

[28] Yang, A.T., *Application of Quaternion Algebra and Dual Numbers to the Analysis of Spatial Mechanisms*, Ph.D. Dissertation, University of Columbia, 1963.

APPENDIX A

DESIGN CHARTS

The design charts developed in the body of this thesis are collected here for more convenient use.

Figure 11: Link Length Curves for RRSC.4d

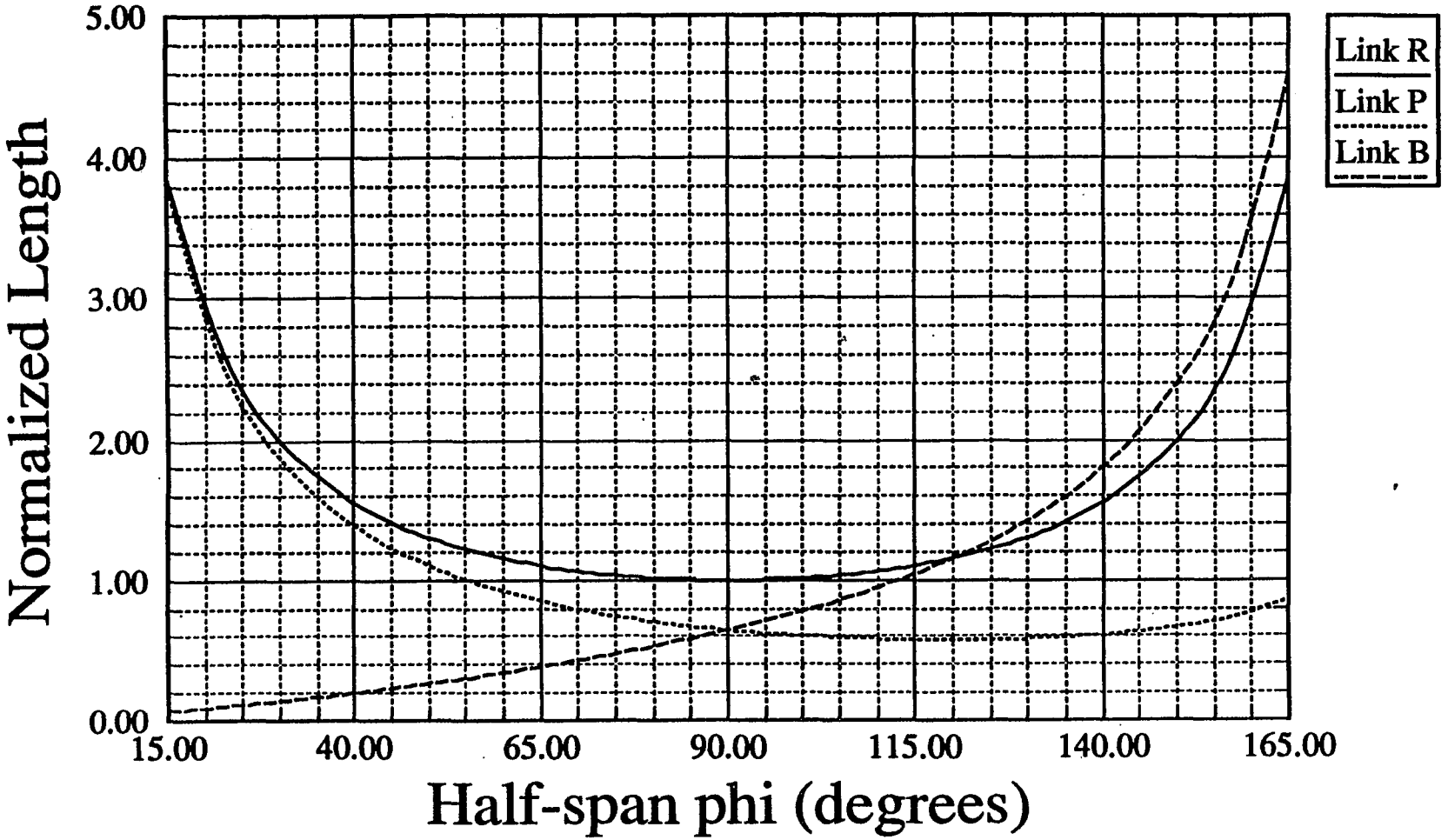


Figure 12: Cylinder Joint Output for RRSC.4d

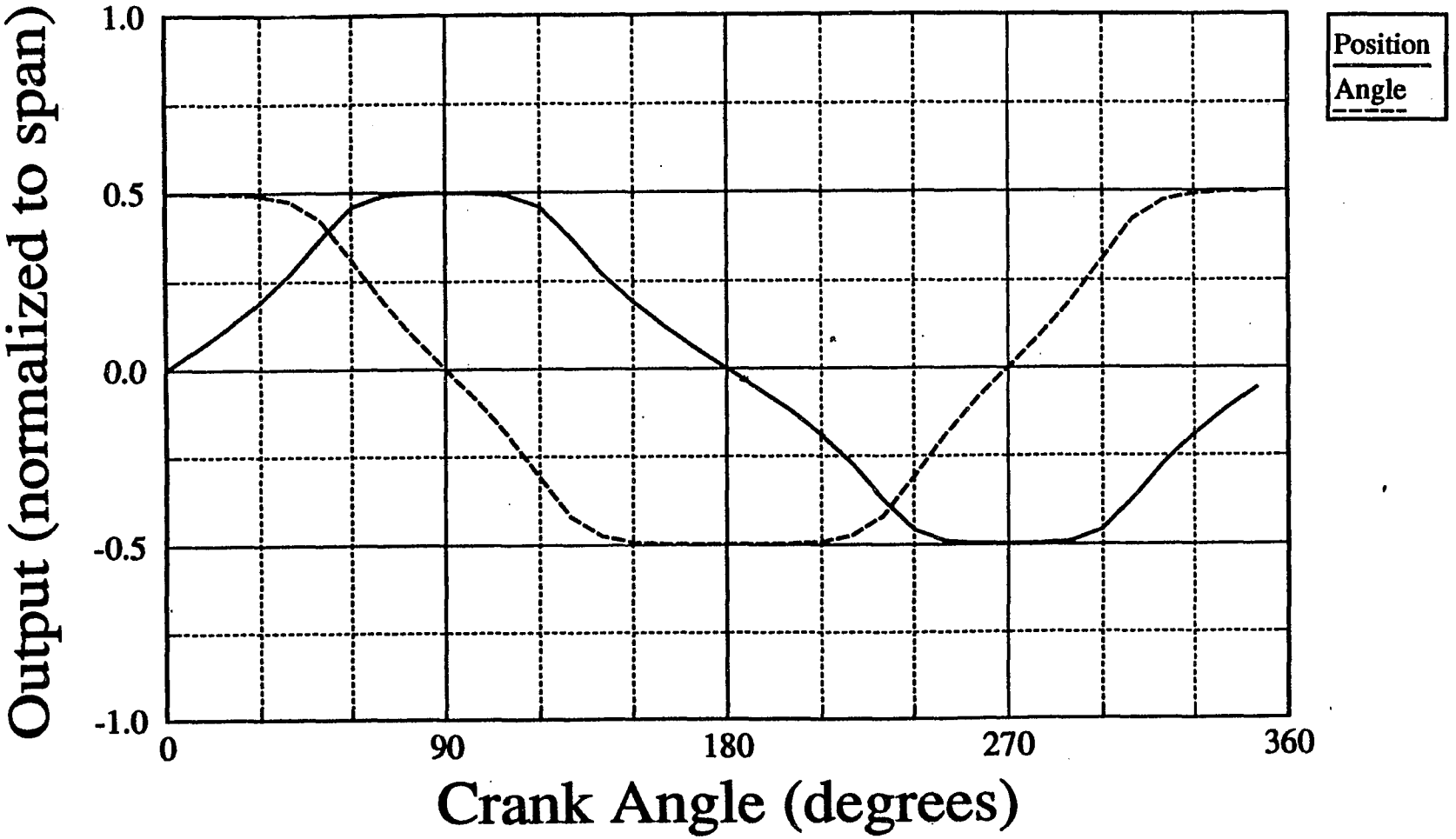


Figure 13: Angular Dwell Ranges for RRSC.4d

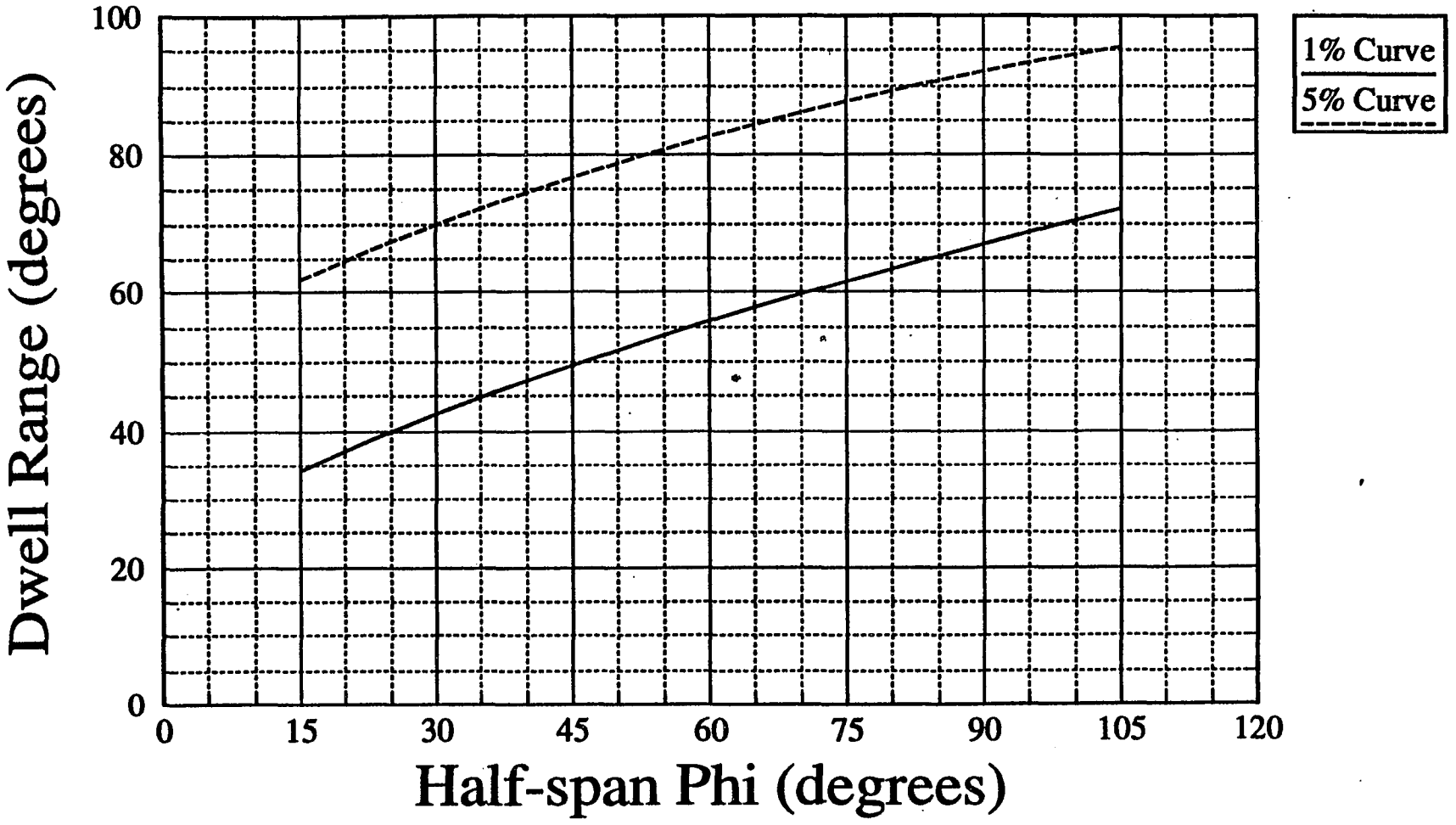


Figure 14: Translational Dwell Ranges for RRSC.4d

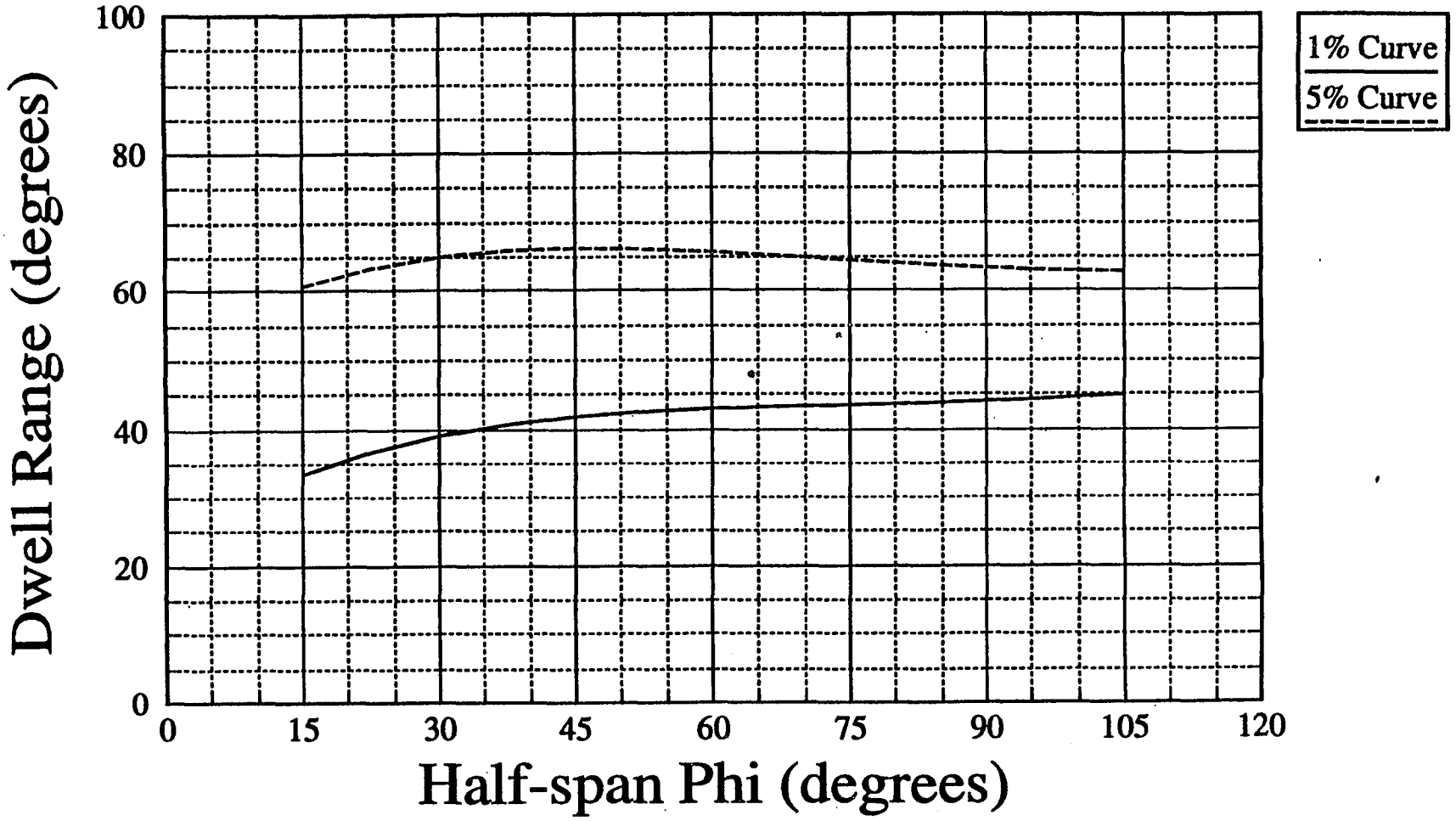


Figure 16: 1% Dwell Ranges for RRSC.2ad

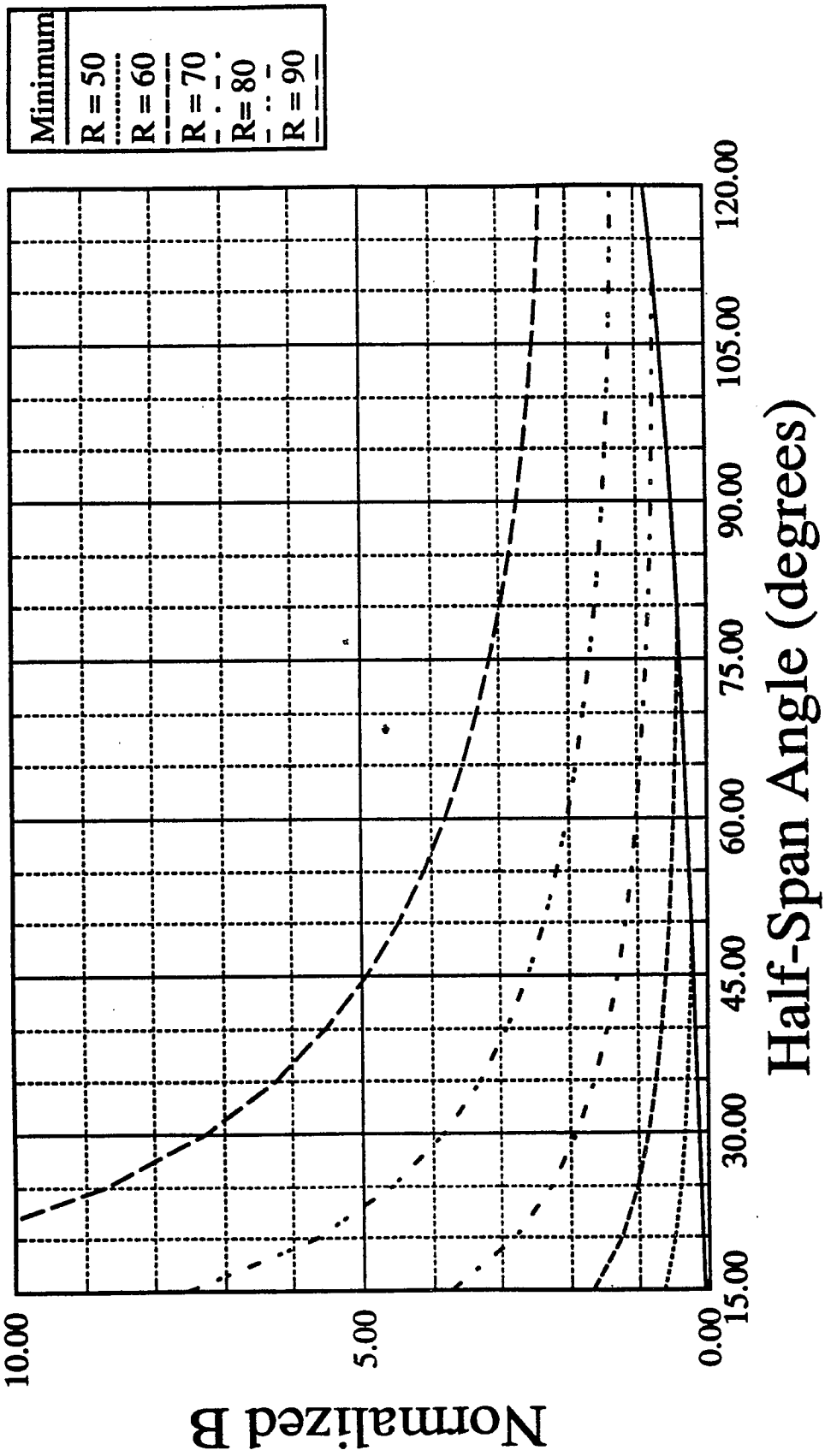


Figure 17: 5% Dwell Ranges for RRSC.2ad

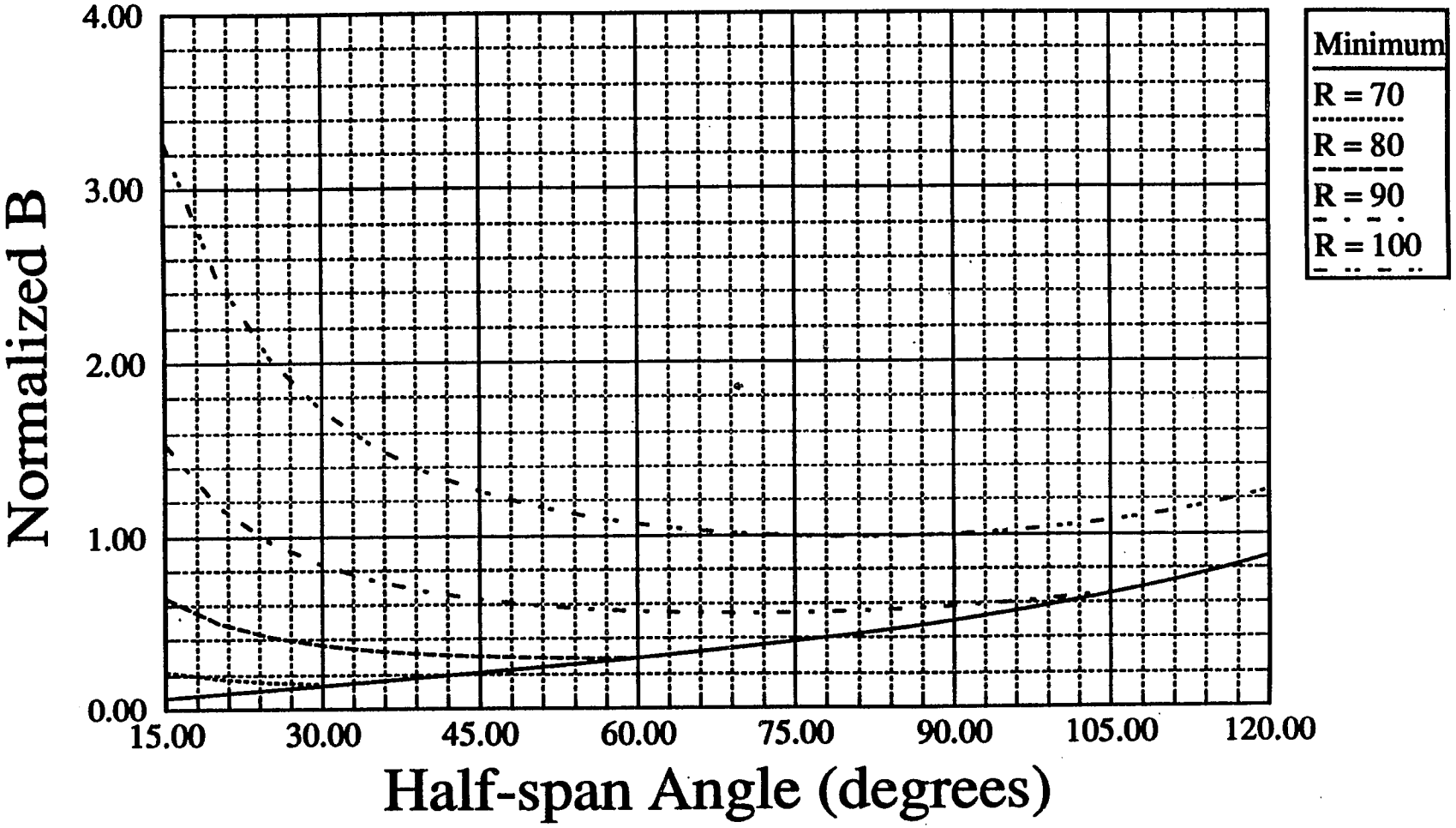


Figure 19: 1% Dwell Ranges for RRSC.2td

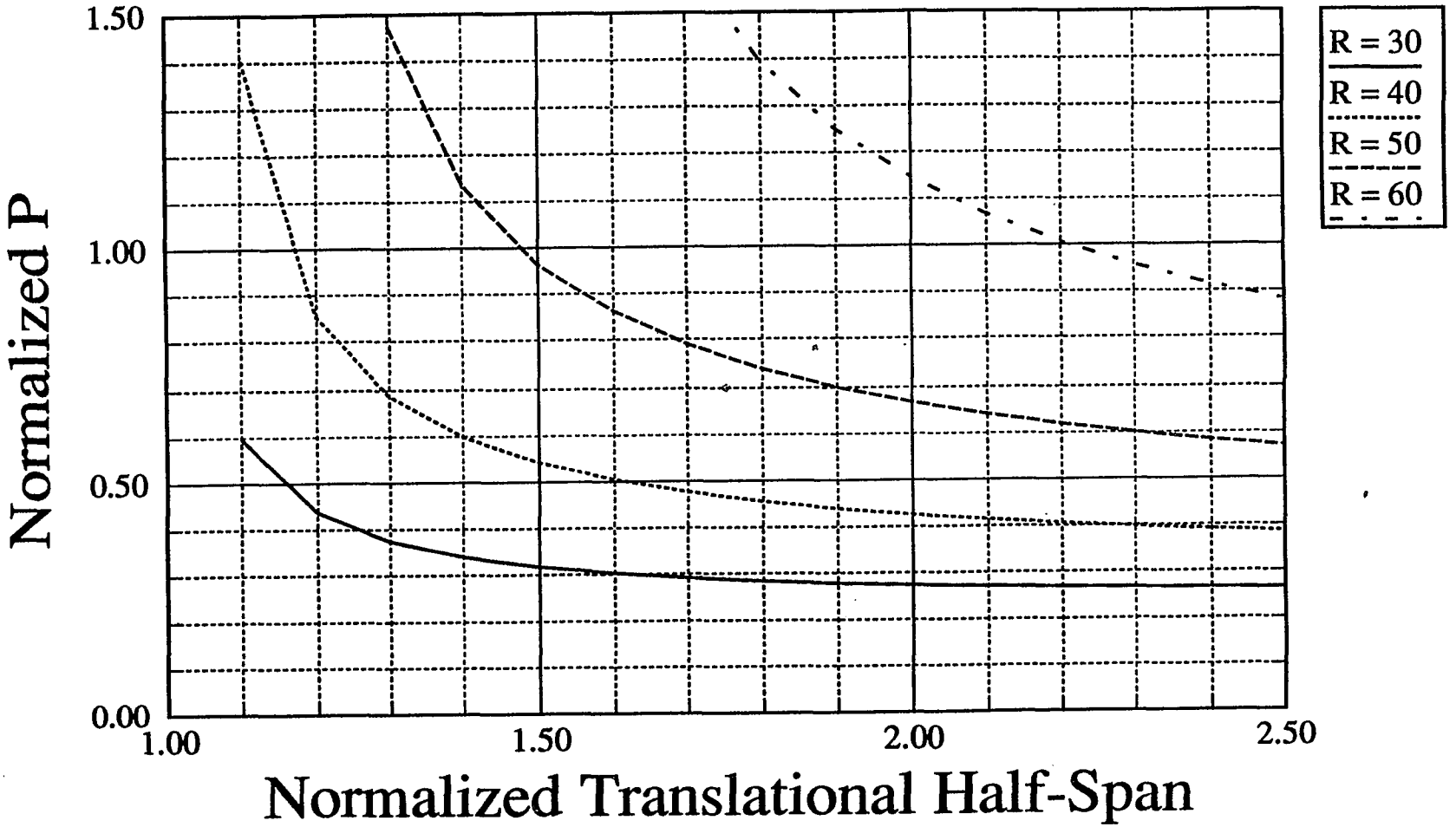


Figure 20: 5% Dwell Ranges for RRSC.2td

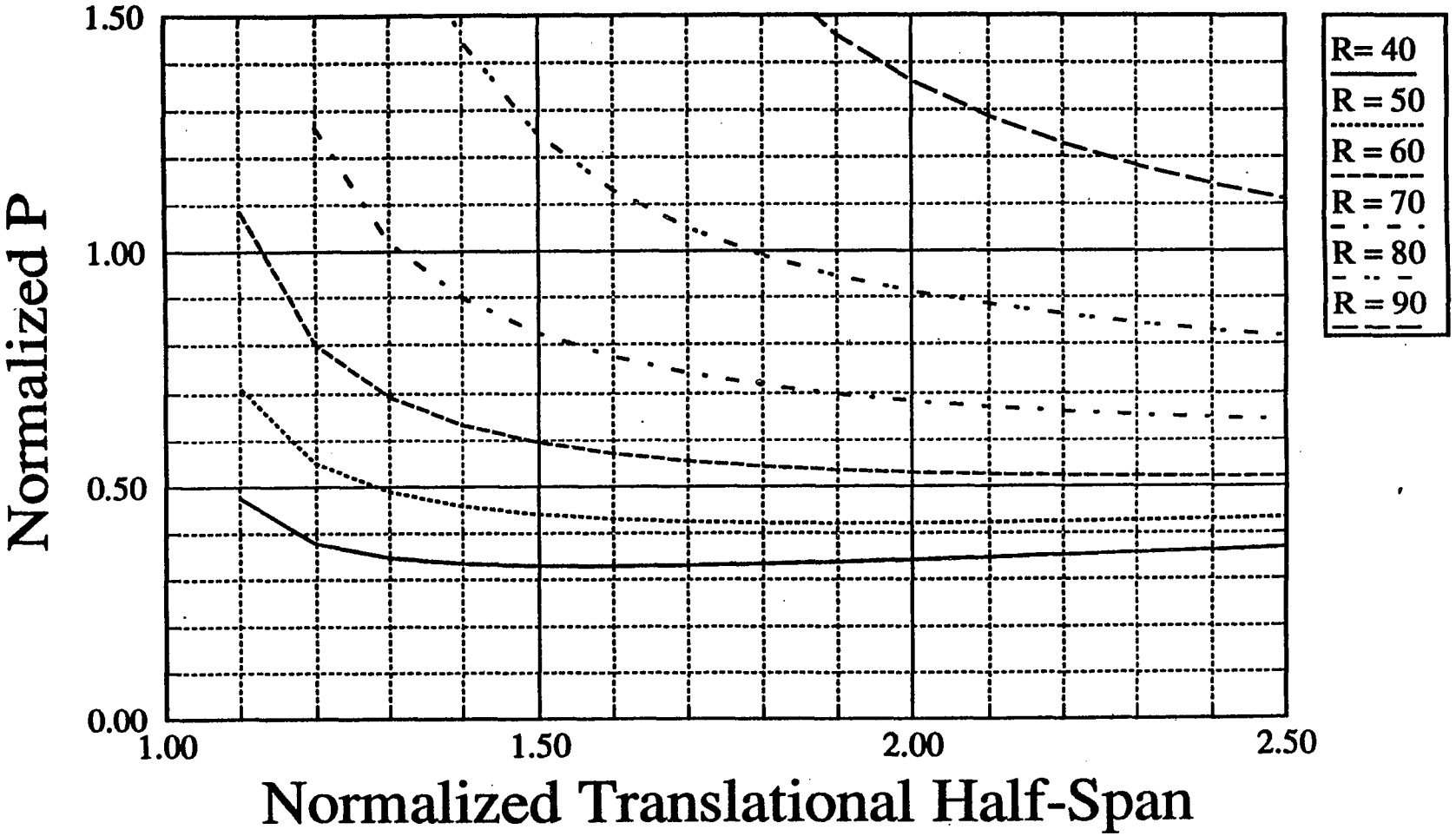


Figure 27: 1% Dwell Ranges for RRSC.1ad (B = 0.5)

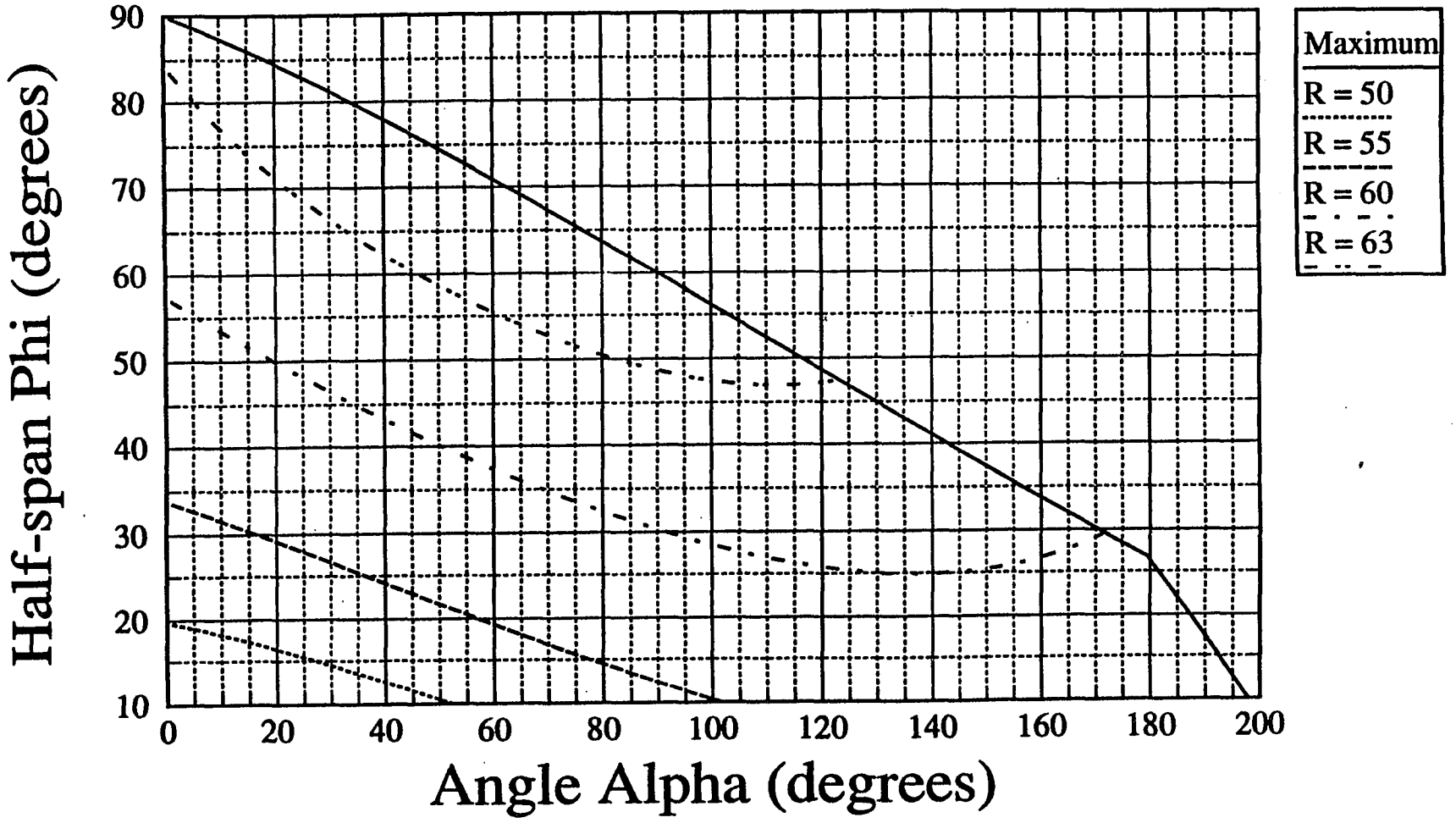


Figure 28: 5% Dwell Ranges for RRSC.1ad (B = 0.5)

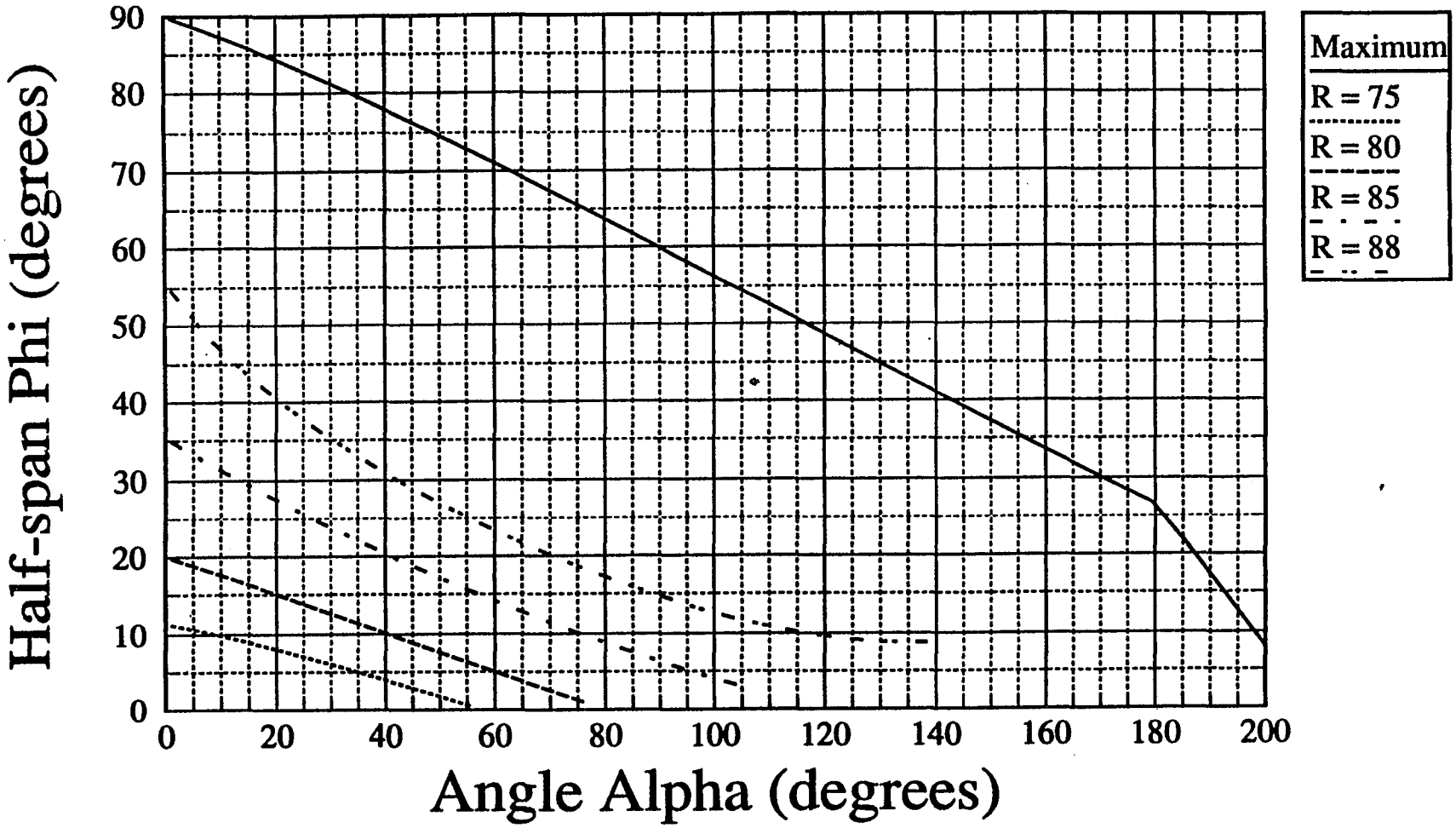


Figure 29: 1% Dwell Ranges for RRSC.1ad (B = 1.0)

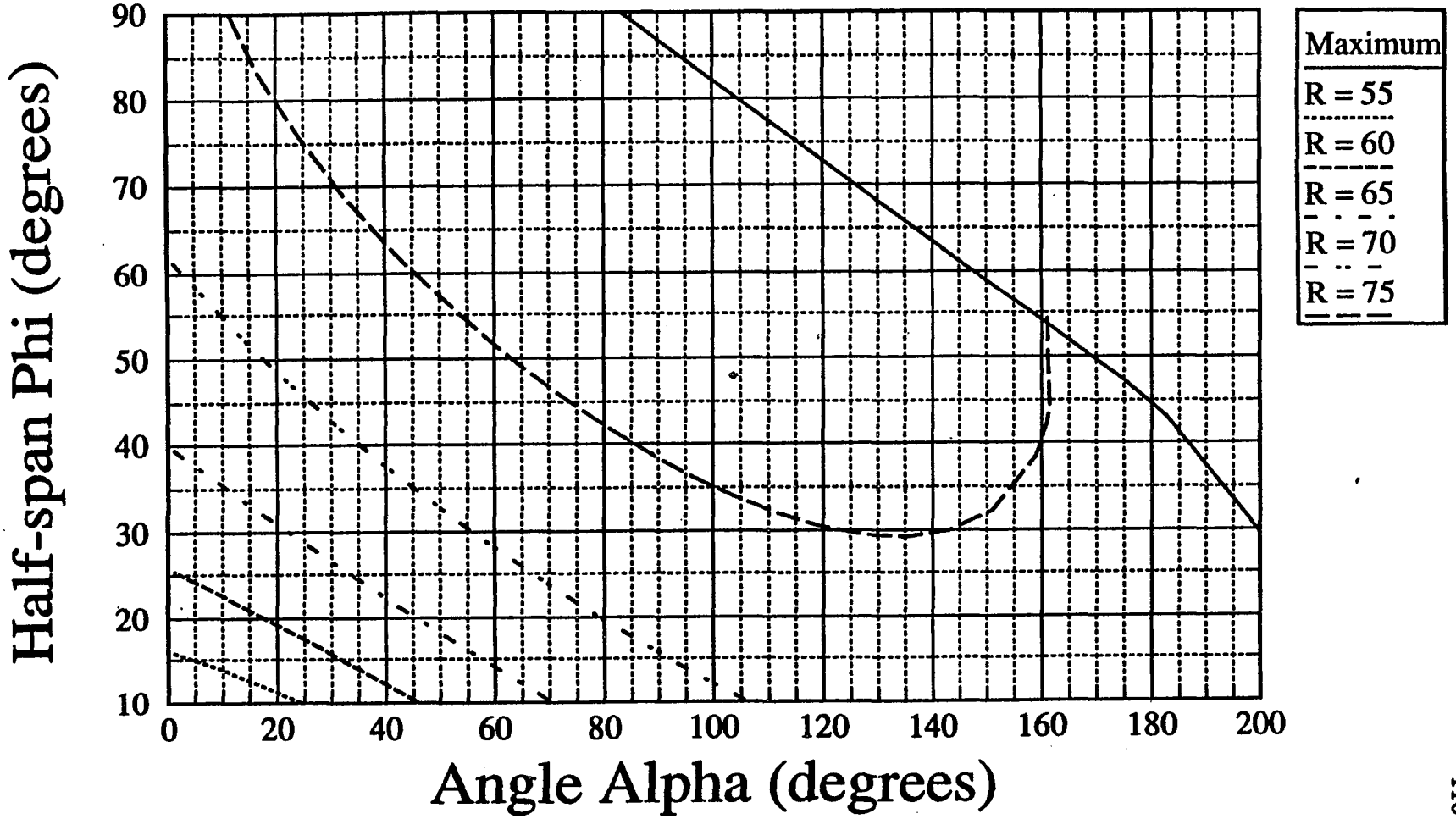


Figure 30: 5% Dwell Ranges for RRSC.1ad (B = 1.0)

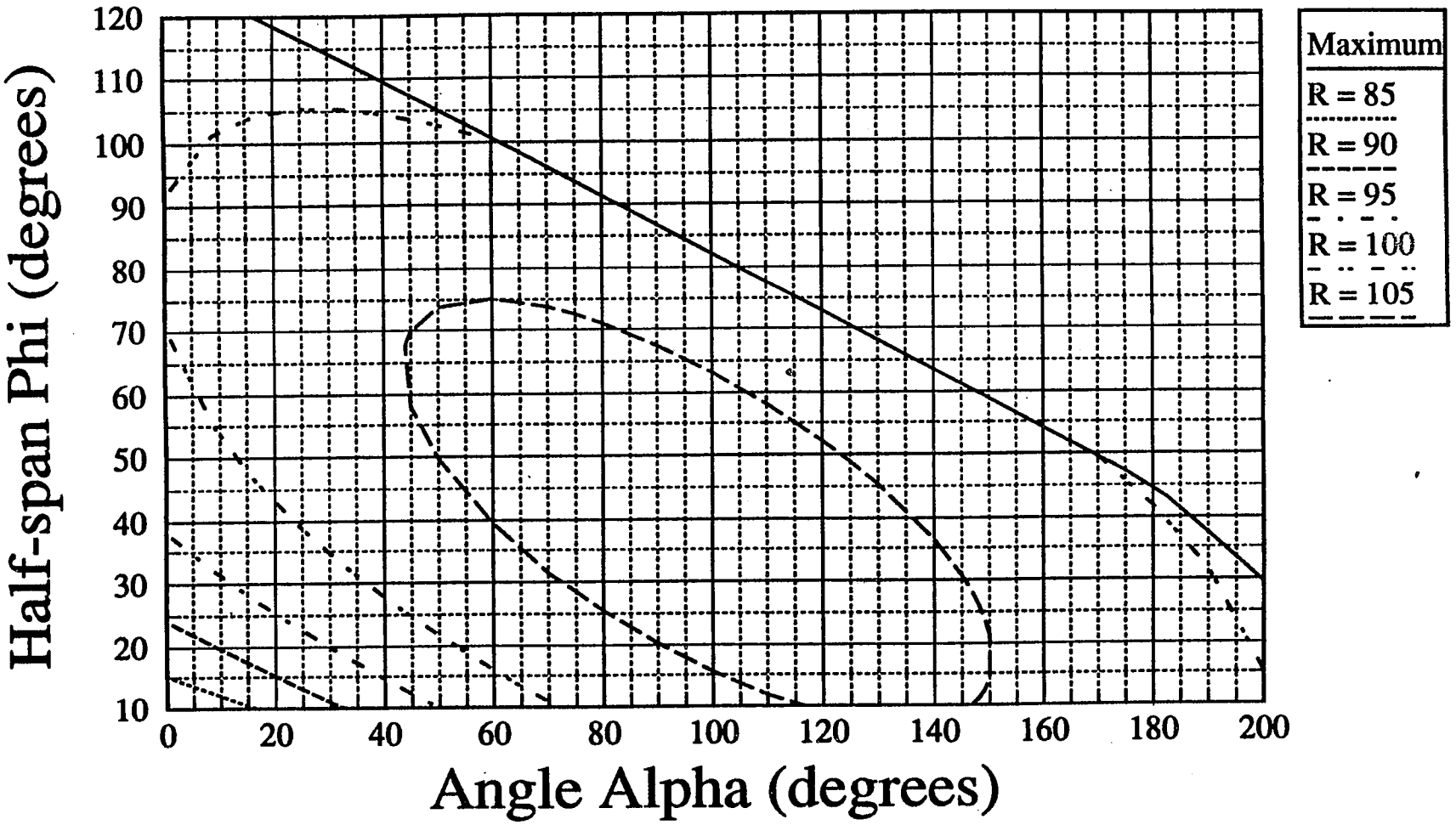


Figure 31: 1% Dwell Ranges for RRSC.1ad (B = 1.5)

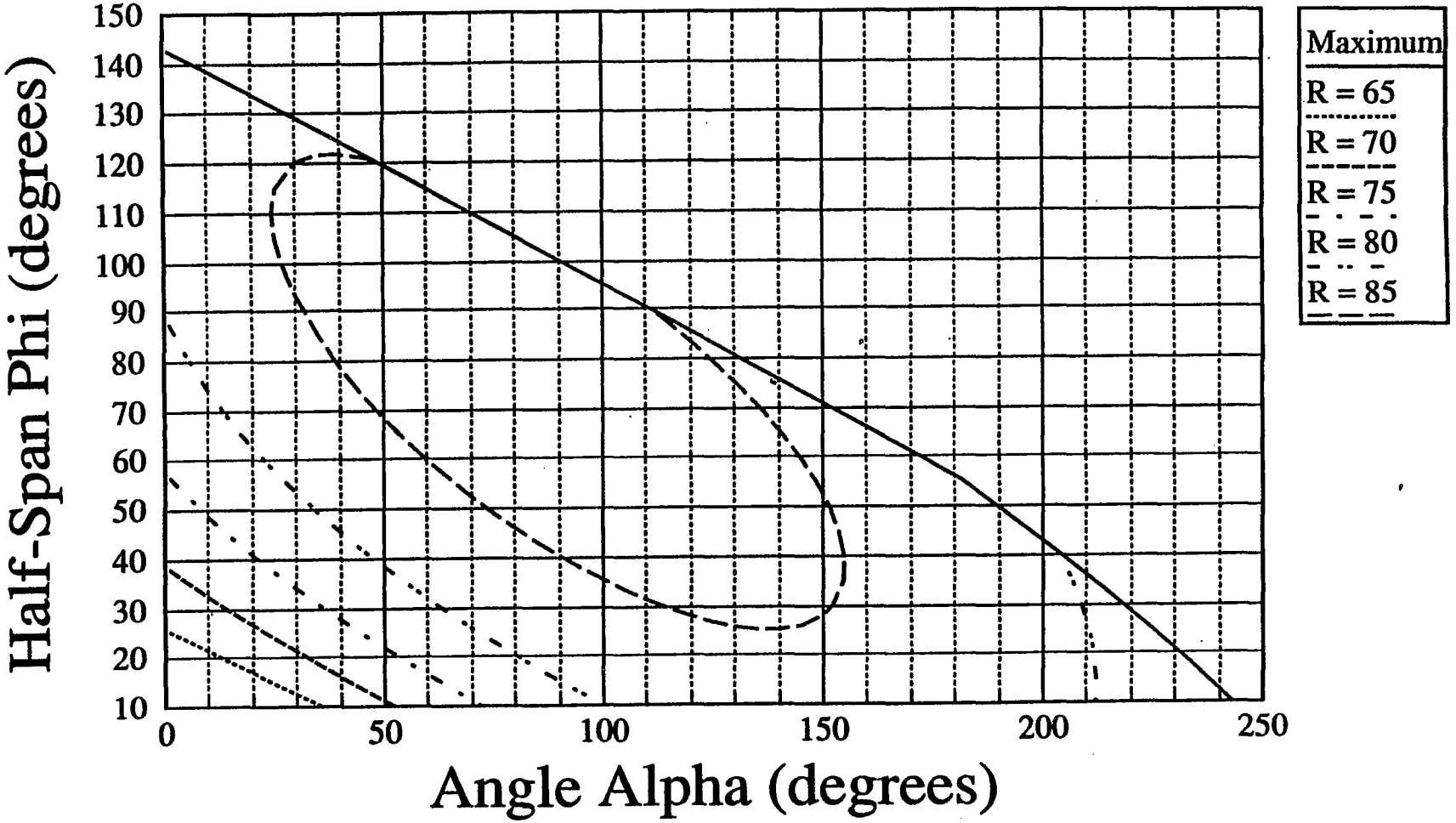


Figure 32: 5% Dwell Ranges for RRSC.1ad (B = 1.5)

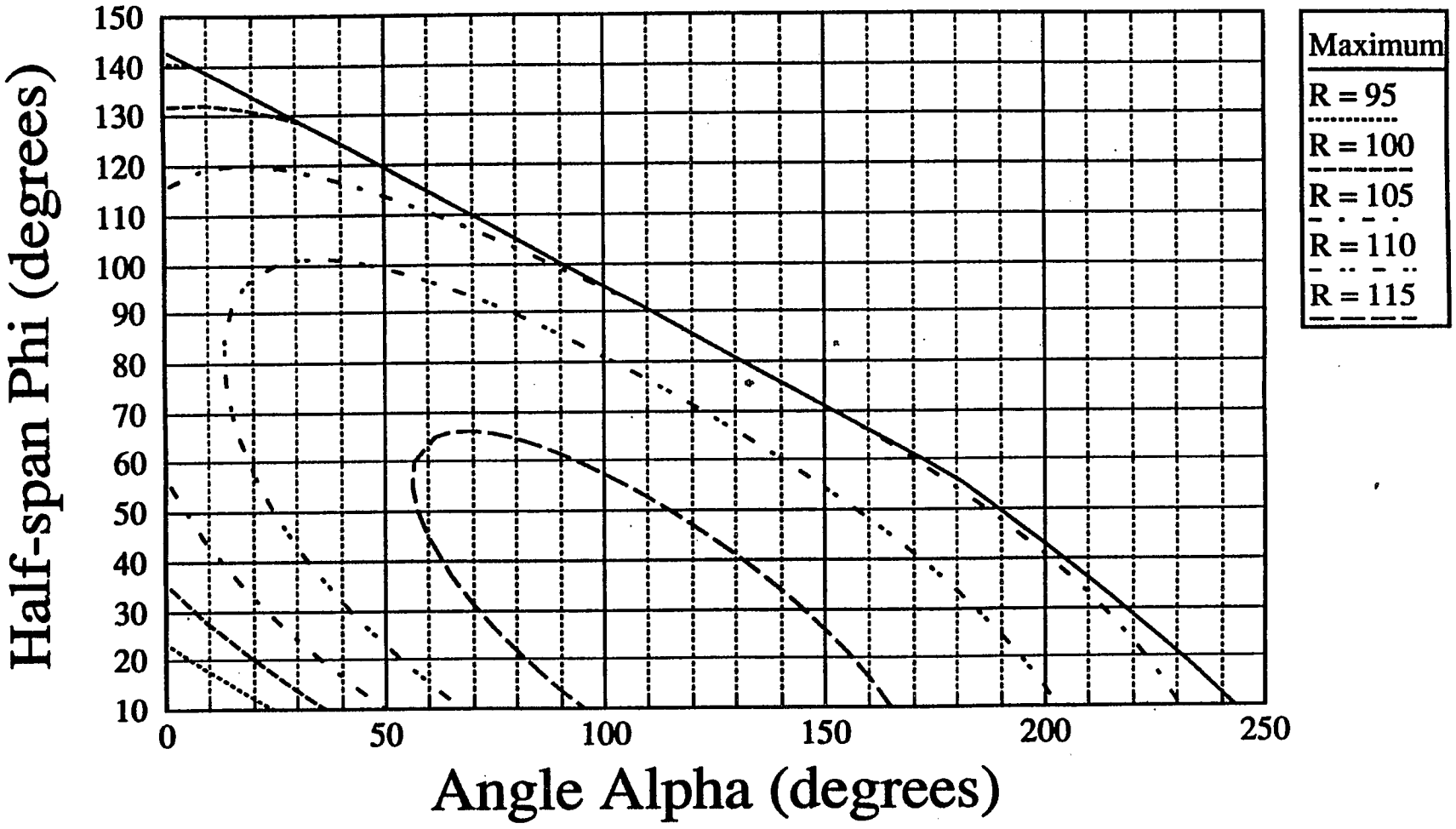


Figure 33: 1% Dwell Ranges for RRSC.1ad (B = 2.0)

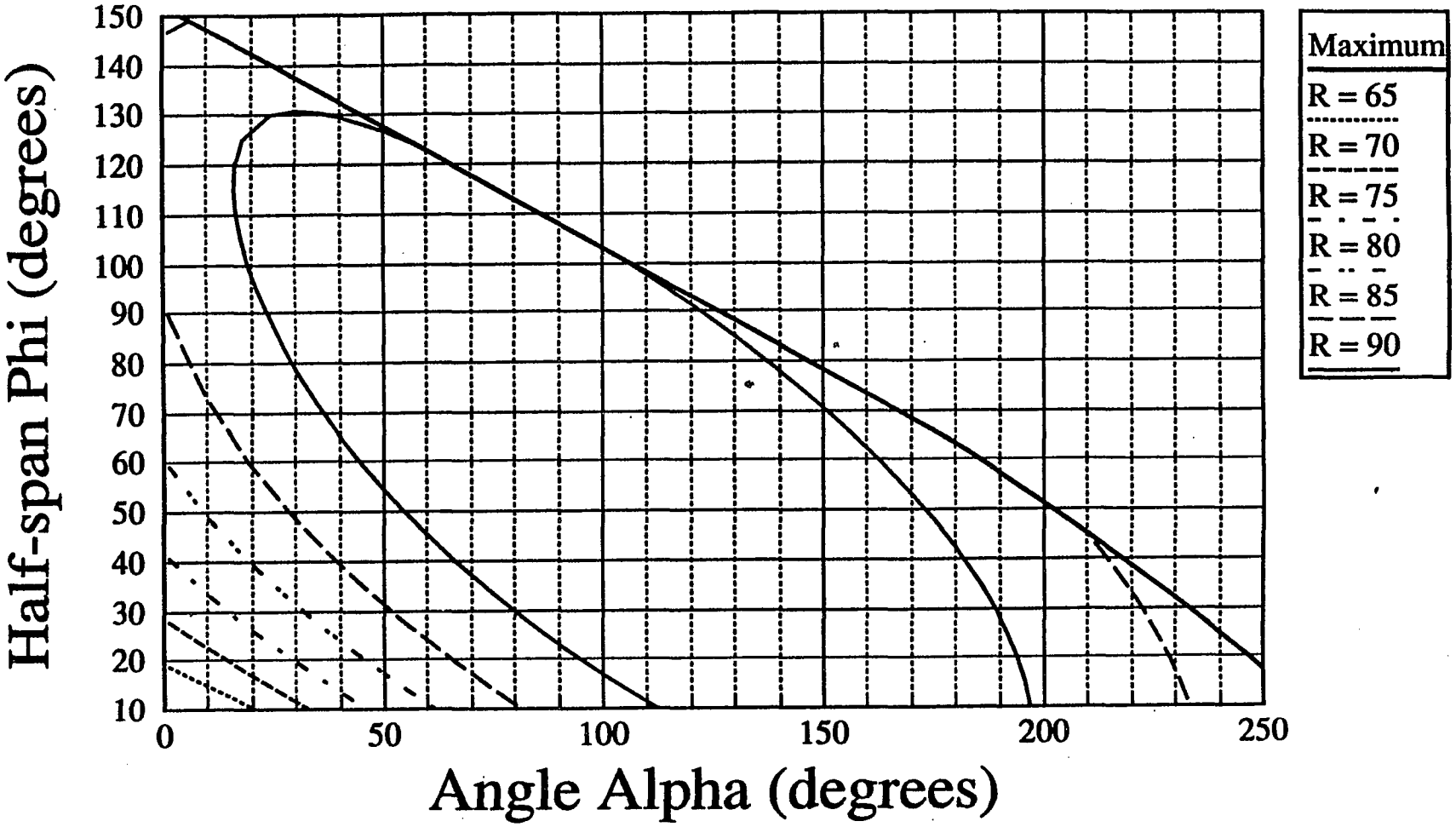


Figure 34: 5% Dwell Ranges for RRSC.1ad (B = 2.0)

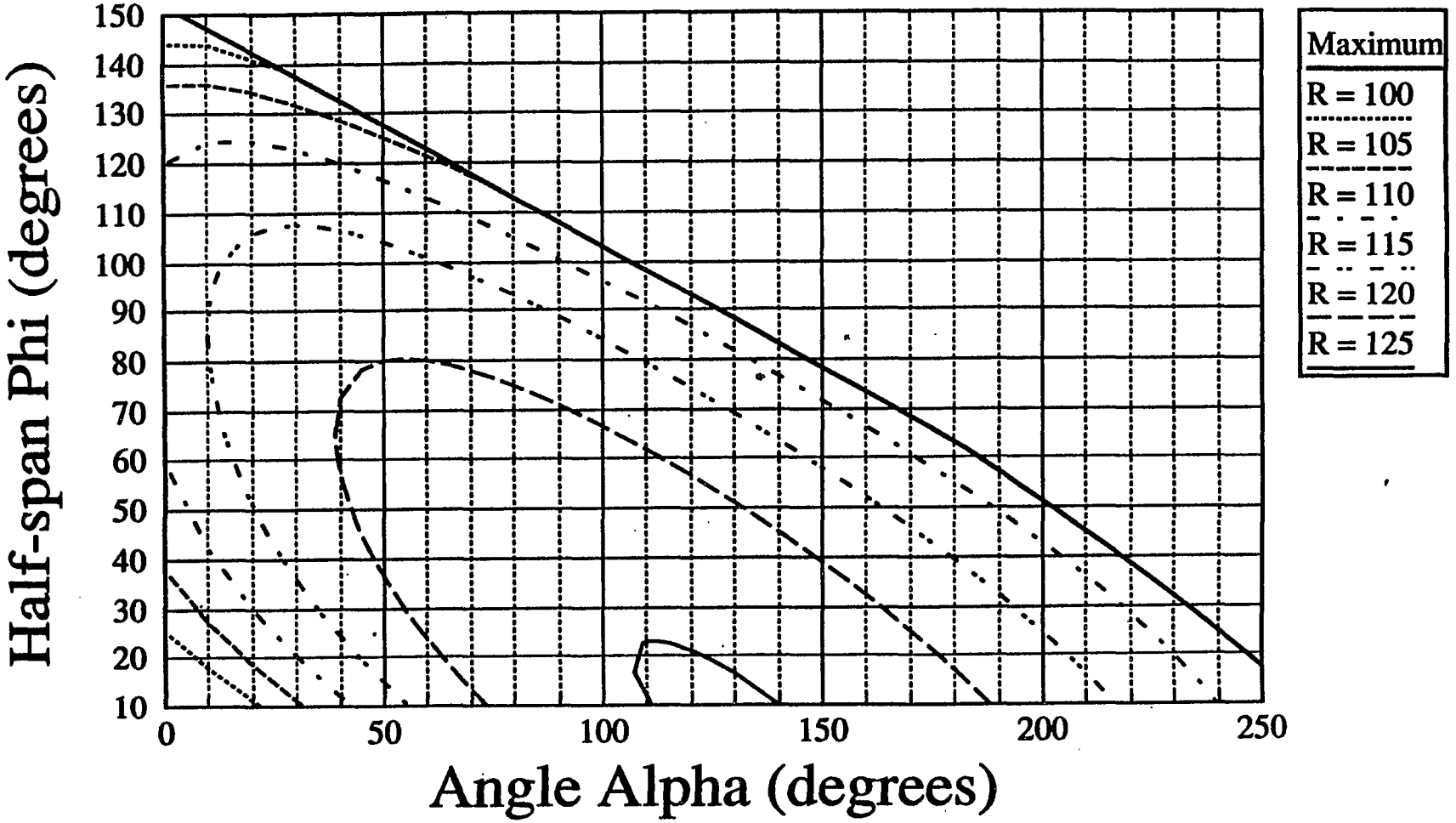


Figure 37: 1% Dwell Ranges for RSRC

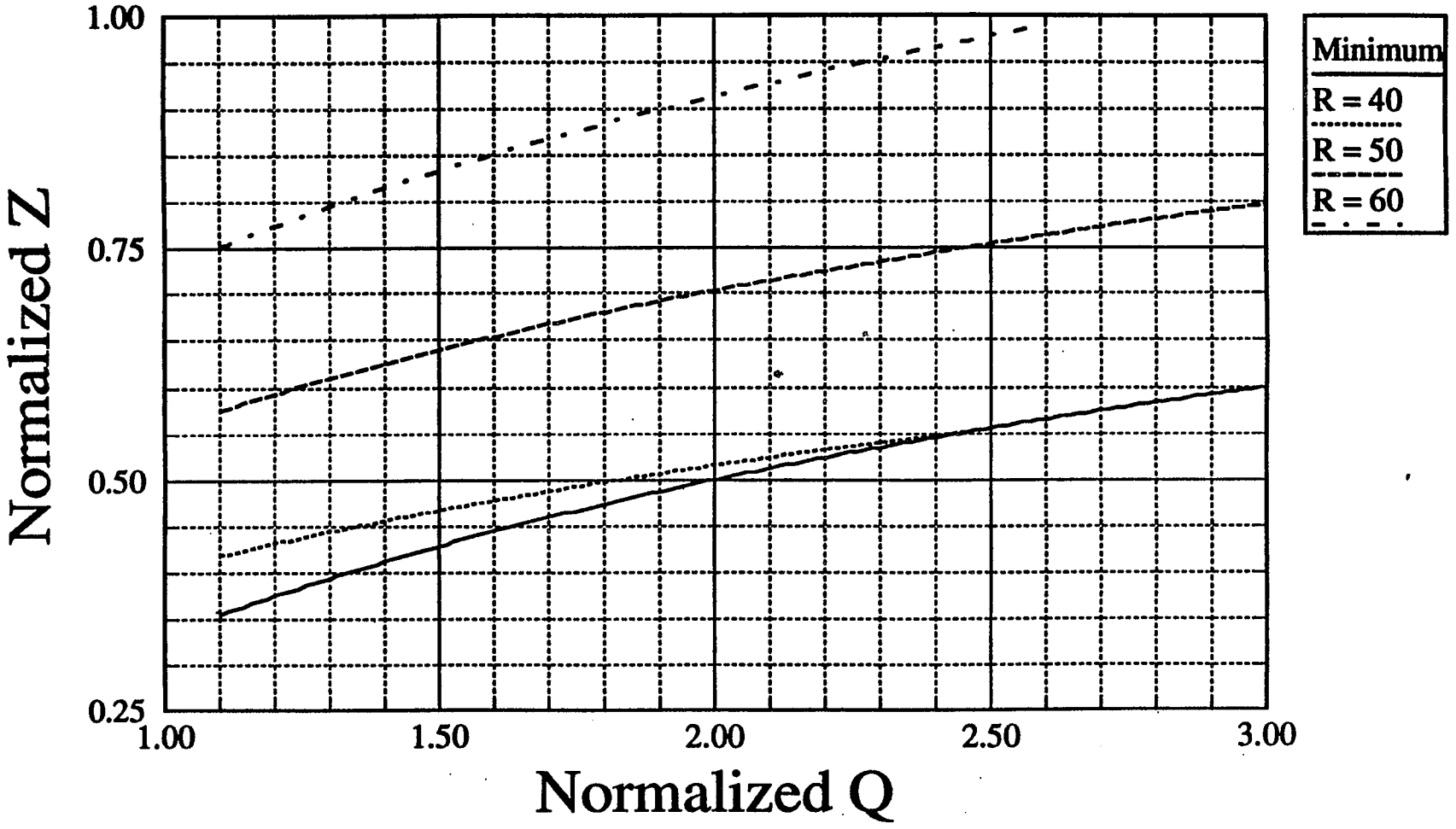


Figure 38: 5% Dwell Ranges for RSRC

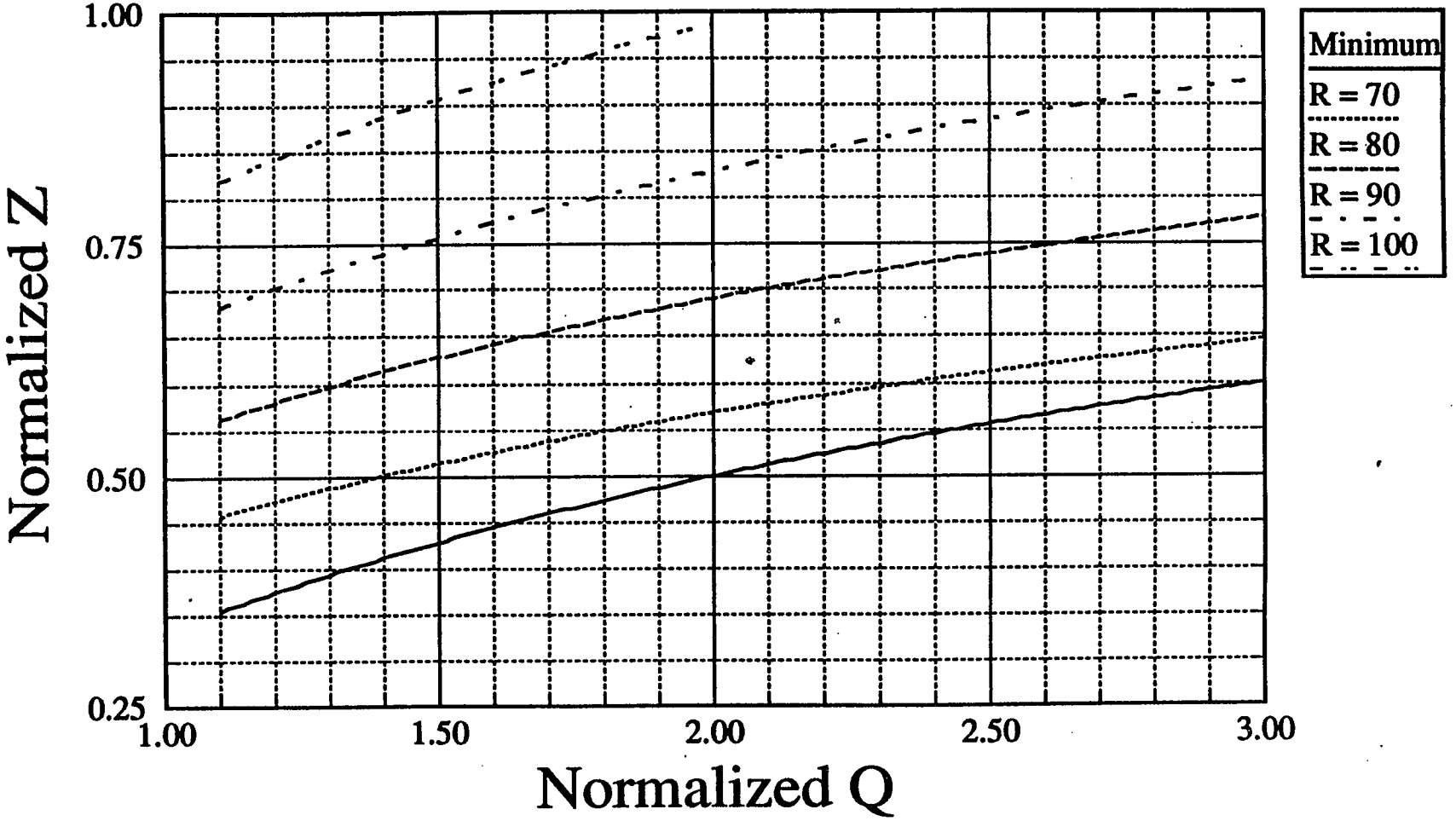
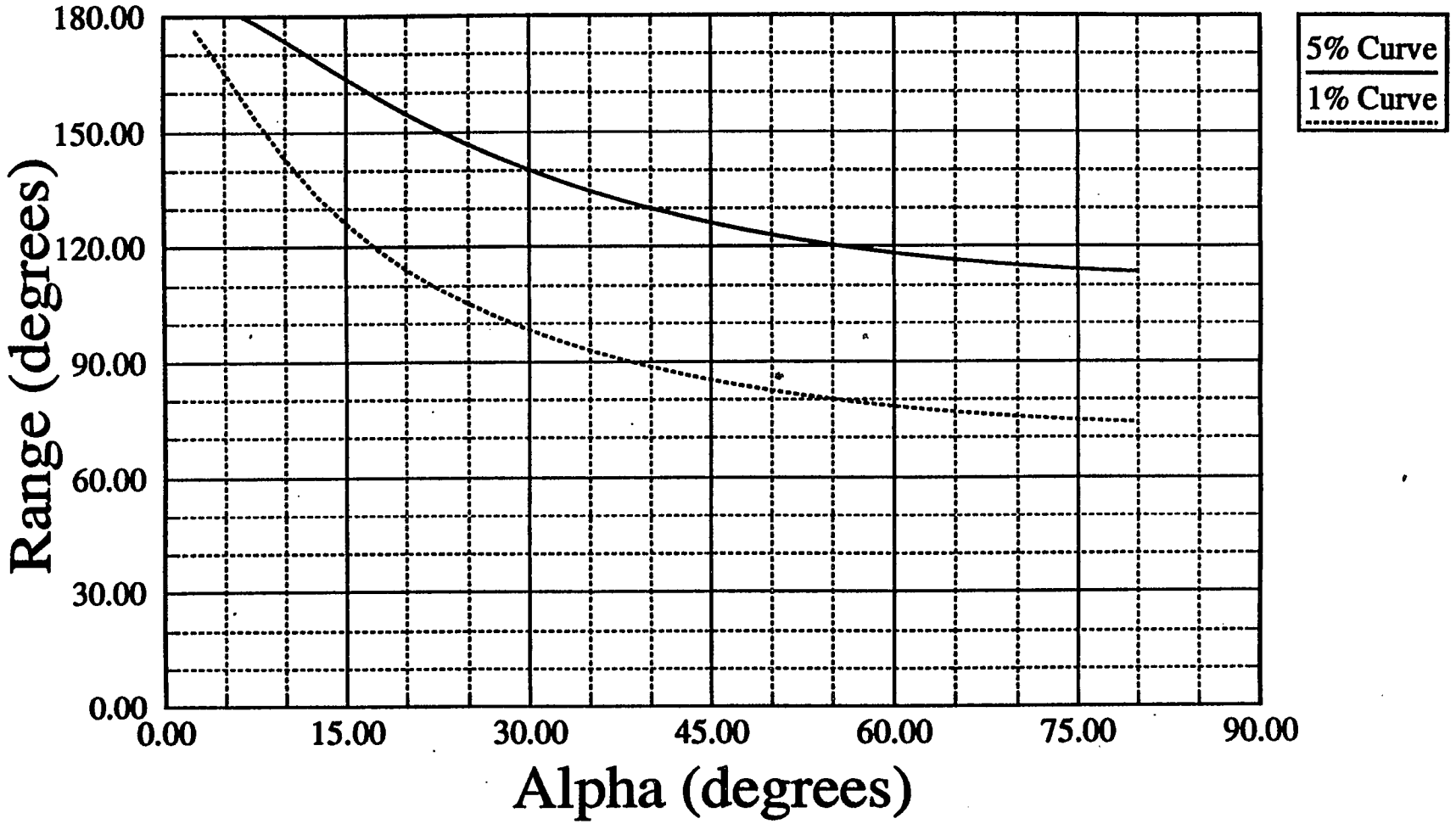


Figure 42: Dwell Ranges for RSCP



APPENDIX B

TK SOLVER MODELS

The following pages contain the TK Solver models used for the linkages studied in this thesis.

B.1 RRSC.4d Link Length Model

St	Input	Name	Output	Unit	Comment
L	35	phi		deg	half span parameter
L	1.7434468	R			link 4 length
L	1.5987812	P			vertical offset
L	.17063315	B			link 3 length
		x	.81915204		dummy variable
		y	.57357644		dummy variable
L		theta6	-55	deg	cylinder rotation
		alpha		deg	crank angle
		beta		deg	R23 angle
		gamma		deg	cylinder rotation
		T			cylinder translation

S Rule

* x = cos(phi)
* y = sin(phi)
* R = 1/y
* P = x/y + B
* (B/R)^3 - (1-2*x)*(B/R)^2 + (1-x+x*x)*(B/R) = (1-x)/2
* theta6 = phi-pi()/2
* cos(alpha) * (1 + B*sin(beta)) = R*sin(gamma)
* B*cos(beta) + R*cos(gamma) = P
* T = sin(alpha) * (1 + B*sin(beta))

B.2 RRSC.2ad Linkage Model

St	Input	Name	Output	Unit	Comment
L	45	phi1			half-span design parameter
L		phi2	.2445948		B design parameter
	1	q			crank length
		r	1.4142136		link 4 length
		p	1.2445948		cylinder height
		b	.2445948		link 3 length
L		alpha	25		crank angle
L		beta	20.562981		R23 angle
L		gamma	44.1		cylinder angle
L		t	.45892569		cylinder translation
	50	range			dwell range
	.01	dev			tolerance on dwell range
		margin	.07497605		margin > 0 for full crank rotation

S Rule

* cosd(alpha)*(q + b*sind(beta)) = r*sind(gamma)
* b*cosd(beta) + r*cosd(gamma) = p
* t = sind(alpha)*(q + b*sind(beta))
* range = 2*alpha
* gamma = (1 - 2*dev)*phi1
* r = 1/sind(phi1)
* p = r*cosd(phi1) + phi2

```
* b = phi2

* margin = 2*phi2 -(1-cosd(phi1))/sind(phi1)
```

B.3 RRSC.2td Linkage Model

St	Input	Name	Output	Unit	Comment
L	1.7	phiT			translational half-span parameter
LG	1.62	phiP			P parameter
	1	q			length of link#2
		b	2.0009476		length of link#3
		r	2.2071796		length of link#4
		p	.88287186		height offset of cylinder
		alpha	60		R12 (crank) angle
LG	136	beta			R23 joint angle
		gamma	39.856485		C41 joint angle
		t	2.45		translational position
	60	range			dwell range
	.01	dev			tolerance on range

```
S Rule

* r=q*phiP*phiT

* p = q*phiP

* b = q*sqrt((phiP^2+1)*(phiT-1)^2)

* cosd(alpha)*(q + b*sind(beta)) = r*sind(gamma)
```

```
* b*cosd(beta) + r*cosd(gamma) = p
* t = sind(alpha)*(q + b*sind(beta))
* alpha = 90 - range/2
* t = (1 - 2*dev)*phiT
```

B.4 RRSC.1ad Linkage Model

St	Input	Name	Output	Unit	Comment
	2	b			link #3 length (parameter)
L	125	alpha		deg	parameter
		tau	5.6712818		parameter
		h	9.7433475		-x offset
		p	10.743347		vertical offset
		r	13.851557		link #4 length
		alphMax	270	deg	maximum value for alpha
		tauMin	-.1026341		minimum value for tau
L	10	phi		deg	half-span angle
		span	20	deg	total span
		mu	50.859984	deg	gamma for crank angle 0
		crank	62.677062	deg	crank angle
		beta	24.917253	deg	R23 joint angle
		gamma	49.859984	deg	output variable
		range	125.35412	deg	dwell range
	.05	tol			dwell tolerance
		good	5.773916		good >= 0 for good linkage
		margin	-1.10821		margin > 0 or h > 1

dum2	-.2455756	used to find tauMin
dum1	2.3927285	used to find tauMin
c	-.5735764	cos(alpha)
s	.81915204	sin(alpha)

S Rule

* h = 0.5*b*(sin(alpha) + tau - tau*cos(alpha))
* p = 0.5*b*(1 + cos(alpha) + tau*sin(alpha)) + tau
* r = sqrt((h + 1)^2 + (p - b)^2)
* alphMax = pi() + 2*atan(b/2)
* tauMin = dum2 / dum1
* span = 2*phi
* phi = atan2(1, tau)
* dum2 = c*(0.5*b*s - 1) - 0.5*b*s*(1 + c)
* dum1 = 0.5*b*c*(c - 1) + s*(1 + 0.5*b*s)
* s = sin(alpha)
* c = cos(alpha)
* cos(crank)*(1 + b*sin(beta)) = r*sin(gamma) - h
* b*cos(beta) + r*cos(gamma) = p
* range = 2*crank
* tol = (mu - gamma) / span
* margin = b - r + p
* mu = atan2(h+1, p-b)
* good = gval()

B.5 RSRC Linkage Model

St	Input	Name	Output	Unit	Comment
L	1.1	phiQ			Q parameter
LG	.85	phiZ			Z parameter 0<Z<1
		q	2		length of link#4
	1	b			length of link#3
		f	1.0128157		height offset of R12
		r	.16372778		length of link #2
		span	.32745555		span of output
L		theta	50		crank angle (input)
L		p	.30693931		X offset of C41 (output)
		dum1	.21048437		dummy variable
		dum2	-.0296058		dummy variable
		dum3	1.0205521		dummy variable
		margin	7.9473976		margin > 0 for proper link age
	100	range			dwell range
	.05	dev			deviation from dwell point
		p0	.32331208		p at dwell point

S Rule

* q = b * phiQ

* f = b*(phiQ - phiZ)

* r = b * (phiQ - phiZ) * sqrt(1 - phiZ^2) / phiZ

* span = 2*r

```
* dum1 = 2*r*cosd(theta)
* dum2 = q^2 + r^2 + f^2 - 2*q*dum3 - b^2
* dum3 = sqrt(f^2 + r^2 * (sind(theta))^2)
* p = 0.5 * (dum1 + sqrt(dum1^2 - 4 * dum2))
* margin  = (q + 1)^2 - f^2 -  r^2
* range = 2*theta
* dev = (p0 - p)/ span
* p0 = q*r/f
```

B.6 RSCP Linkage Model

St	Input	Name	Output	Unit	Comment
L	2.5	alpha			parameter angle
		B	1.0009527		link 3 length
		theta	92.382246		crank angle
		D	.10180929		output (displacement)
		x	-.0415265		dummy variable
		y	.99913576		dummy variable
		span	1.9980964		span of output
LG	184.76449	range			dwell range
	.05	dev			tolerance on dwell range
		Dmin	.00190446		minimum displacement

```
S Rule
* B = 1/cosd(alpha)
* D = sqrt(B^2 - y^2) - x
```

```
* x = cosd(alpha)*cosd(theta)

* y = sind(theta)

* span = 2*cosd(alpha)

* range = 2*theta

* Dmin = B - span/2

* dev = (D - Dmin)/span
```

APPENDIX C

EXPLANATION AND CODE FOR SPACELINKS PROGRAM

The linkages studied in this thesis were all modeled and analyzed using the homogeneous coordinate transformation matrix technique mentioned in the Introduction; the purpose of this Appendix is to explore this technique more fully, especially regarding position analysis, and how the method was used in the SpaceLinks linkage analysis program. The use of SpaceLinks for modeling and analyzing linkages will also be explained.

The modeling of a linkage starts with the modeling of its joints. Each joint is represented by a concatenation of transformation matrices, one for each joint variable or degree of freedom; the linkage itself is represented by the concatenation of the joint models. If the joint variables have their proper values, the total transformation matrix should be equal to the 4x4 identity matrix. This is equivalent to the equation S - I = 0, where S is the transformation matrix for the entire model (and thus is a function of all the joint variables), I is the identity matrix, and 0 is the 4x4 zero matrix. This is a root-finding problem, and the technique used here is based on the well known Newton-Raphson method for finding the roots of an equation.

The Newton-Raphson method is basically iterative: given an initial guess of the root, a closer approximation is found. This method converges rapidly, provided that the initial guess is close to the root. For a scalar function of only one variable, the solution x to f(x) = 0 is found from the iterative formula x := x - f(x)/f'(x); for a vector function of a vector variable the iterative formula for $\mathbf{x}$: $\mathbf{F(x) = 0}$ is $\mathbf{x := x - y}$ where

$J(\mathbf{x})\ \mathbf{y} = F(\mathbf{x})$ and J is the Jacobian of **F**. For the solution of $S(\mathbf{q})$-I=0, the method is based on the iterative formula

$$\mathbf{q} := \mathbf{q} + \Delta\mathbf{q} \text{ where } S(\mathbf{q}) - I + \Sigma\Delta qi\partial S/\partial qi = 0.$$

The SpaceLink application was written in THINK C, using Nicus' NuTools: Numerical library of numerical subroutines, and is based on a set of data types to hold the model, and a set of subroutines to manipulate these data types. Using these data types and "code primitives," other subroutines create and analyze the model, and finally a Macintosh "application shell" controls program flow, file I/O, and the user interface. This explanation focuses mainly on the data types and code primitives; the application shell will not be discussed.

The first data type used in the code primitives is the NuTools matrix data type. This is used to hold all matrices in the various routines; also, one column matrice holds all the joint and link length variables associated with the model. The other data type definition is for a type called "data" in the program. This consists of a structure of six integers, which hold the information needed for one transformation. The first field holds the type of elementary transformation represented, and the next five fields hold the positions in the variable matrix of the relevant variables for this transformation. In the program, a block of memory is dynamically allocated to hold all the transformations, and pointers to the transformations are what is usually passed as a parameter.

Of the analysis subroutines, the most basic is the "transform()" function, which performs a matrix multiplication corresponding to the transformation desired. This function is called by the "get_s()" function, which returns a matrix corresponding to a

concatenation of transformations; the "invert_s()" function inverts this concatenation matrix. The "get_r()" function calls "get_s()" and "invert_s()," and the "get_d()" function uses the results returned by "get_s()." These two functions return intermediate column matrices used to solve for $\Delta\mathbf{q}$. The above four interdependent functions are the "code primitives" called by the "iterate()" function to update the $\mathbf{q}$ column matrix; "iterate()" also returns the infinity norm of $\Delta\mathbf{q}$ as a measure of how accurate the new approximation is.

In the SpaceLinks program, the "iterate()" function is called in a loop which terminates when the desired accuracy is reached or when the Newton-Raphson fails to converge.

C. 1 Use of the SpaceLinks Program

The SpaceLinks application shell is based on the Macintosh interface. Program control is accomplished through menus for file input and output, and position analysis and display. The types of analysis available are position analysis of a single linkage, batch processing of several position analyses, and batch processing of analysis of position changes from a reference position.

There is no provision for interactive input of linkage models; SpaceLinks can only read previously prepared linkage files. These files are character-based files of type 'TEXT' and so they can be prepared on any word processor. Since SpaceLinks supports Desk Accessories, these text files may be prepared by DA text editors, such as MockWrite and miniWRITER, without exiting SpaceLinks.

The first task in developing a SpaceLinks model is the choice of the transformations to represent the linkage. Several rules must be followed for a successful choice of transformations:

I. There is at least one transformation for each joint variable.

II. The concatenation of transformations must return to the original position and orientation.

III. Only certain transformations are legal in SpaceLinks. There are nine legal transformations:

1. Translation followed by rotation about X-axis.
2. Translation followed by rotation about Y-axis.
3. Translation followed by rotation about Z-axis.
4. Translation followed by screw motion along X-axis.
5. Translation followed by screw motion along Y-axis.
6. Translation followed by screw motion along Z-axis.
7. Translation, followed by translation along X-axis.
8. Translation, followed by translation along Y-axis.
9. Translation, followed by translation along Z-axis.

IV. In transformations involving rotation (transformations 1 - 6), only the rotation angle may be a joint variable. All other parameters of the transformation must be constants.

Once the transformations have been chosen, the joint variables and other transformation parameters must be listed. These are placed in the following order: first come the dependent joint variables, then the independent variable (such as crank angle, or other linkage input), any indeterminate joint variables, and finally all constant transformation parameters such as link lengths. These are numbered starting from zero to form the components of the **q** column matrix. Values for all of these components must be known at least approximately for the model in its first position.

The text file is written from the above data. The first line contains the number of transformations and the number of parameters, and the second line contains the number of dependent variables and the position of the independent variable in the parameter vector.

The next lines contain information on the transformations, one transformation per line. Each transformation is represented by six integers:

1. The type of transformation (1 - 9).
2. Position of X-translation component in parameter vector.
3. Position of X-translation component in parameter vector.
4. Position of X-translation component in parameter vector.
5. Position of rotation angle in parameter vector.
6. Position of screw parameter in parameter vector.

A negative integer is used to represent transformation components that do not apply to a particular transformation, for example, the sixth integer for Transformation Type 1 may be represented by -1.

The next line contains the number of transformation parameters, followed by the number one. The final lines of the model contain the values of the parameter vector. Throughout the text file, the numbers may be separated by any number of any whitespace characters (such as blank spaces or tabs).

C.1.1 Example Model: RRSC The RRSC mechanism was modeled and studied using SpaceLinks. The model is developed here. There are seven transformations:

1. Rotation of θ_1 about Z-axis.
2. Translation (1, 0, 0) followed by rotation of θ_2 about Y-axis.
3. Translation (0, B, 0) followed by rotation of θ_3 about Z-axis.
4. Rotation of θ_4 about Y-axis.
5. Rotation of θ_5 about X-axis.
6. Translation (-R, 0, 0) followed by rotation of θ_6 about Y-axis.
7. Translation (0, -T, -P), where P is considered constant.

There are eleven transformation parameters:

0. Negative of Translational Output: -T (C_{41} translational variable).

1. θ_2 (R_{23} variable).

2. θ_3 (S_{34} variable).

3. θ_4 (S_{34} variable).

4. θ_5 (S_{34} variable).

5. θ_6 (C_{41} rotational variable).

6. θ_1 (R_{12} variable - independent variable).

7. Link 2 Length: Q = 1.

8. Link 3 Length: B.

9. Negative of Link 4 Length -R.

10. Negative of Vertical offset: -P.

The linkage model is given below, with the values of the transformation parameter matrix, for RRSC.1ad with phi = 90° and theta1 = 0°, included:

```
7 11
7 6
6
3 -1 -1 -1 6 -1
2 7 -1 -1 1 -1
3 -1 -1 8 2 -1
2 -1 -1 -1 3 -1
1 -1 -1 -1 4 -1
2 9 -1 -1 5 -1
8 -1 0 10 -1 -1
1  11
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.648 -1.0 -0.648
```

The program code follows.

C.2 linkage.h

```
/*   header file for the linkage program */
/*   by Don Kelly, January 1990 */

typedef struct {
     int       n;
     int       dx;
     int       dy;
     int       dz;
     int       ang;
     int       lead;
     } data;

void          f_setup(FILE *theFile, matrix *var, data **p, long *error);
void          setup(matrix *var, data **p, long *error);
void          transform(matrix *s, matrix *var, data *p, long *error);
void          get_s(int k, matrix *s, matrix *var, data *p, long *error);
void          invert_s(matrix *s, matrix *sinv, long *error);
void          get_r(matrix *r, matrix *var, data *p, long *error);
void          get_d(matrix *s, matrix *d, matrix *var, data *p, long *error);
double        iterate(matrix *var, data *p, long *error);

void          WindowInit(void);
void          AdjustMenus(void);
void          BuildMenus(void);
void          IsWNEImplemented(void);
void          bye(void);
void          do_menu(long mResult);
void          Loop(void);
Boolean       IsDAWindow(WindowPtr whichWindow);

void          MoveToNextLine(FILE *theFile);
void          GetNewLinkage(matrix *var, data **p, long *error);
void          DumpPositionAnalysis(FILE *outFile, matrix *pos, long *error);
void          WriteOutput(FILE *theOutFile, double firstVar, double secondVar, long
              *error);
void          CloseLinkage(void);
Boolean       ReadNextTrial(matrix *var, char *outFileName, double *span, long
              *outVar, long *error);

void          ViewCurrentPosition(matrix *thePos);
void          ViewPositionAnalysis(void);
void          ScrollMyWindow(void);
void          DrawMyMessage(char *myMessage);
void          SayFirstMessage(void);
void          SetSolverControl(int *steps, double *increment, double *runLength);
void          UpdateScreen(Boolean updateFlag);
```

```
void        WriteMessage(char *messageString);

void        InitSolver(void);
void        RunSolution(void);
void        FindPosition(void);
void        TempWrite(void);
void        ListProcess(void);

#ifndef _THIS_IS_MAIN_
    extern FILE     *gTheInFile;
    extern int          nvar, ndep, ntran, pindep;

    /* OTHER GLOBALS (MENUS, EVENT RECORDS, ETC.) TO GO HERE */
    extern MenuHandle   gMyMenus[7];

    extern WindowPtr    gMyWindow;
    extern int          gMaxRow, gCurRow;
    extern Rect         gDragRect;

    extern EventRecord  gTheEvent;
    extern Boolean      gWNEImplemented;

    extern matrix       gVar, gPositionAnalysis;
    extern data         *gTheLinkage;
    extern long         gErrorFlag;
    extern unsigned int gStatusFlag;
    extern int          lastWritten;
    extern int          nextSent;
    extern char         *messageList[15];
#endif

/* DEFINES FOR MENU BARS, ITEMS, AND OTHER RESOURCES */

#define BASE_RES_ID
    400
#define appleID         400
#define fileID          401
#define editID          402
#define runID           403
#define viewID          404
#define continueID      100
#define optionsID       101

#define appleM          0
#define fileM           1
#define editM           2
#define runM            3
#define viewM           4
#define continueM       5
#define optionsM        6
```

```
#define aboutItem       1

#define openItem        1
#define closeItem       2
#define saveLItem       3
#define saveKItem       4
#define quitItem        5

#define undoItem        1
#define cutItem         3
#define copyItem        4
#define pasteItem       5
#define clearItem       6

#define posItem         1
#define listPosItem     2
#define listDevItem     3
#define stopItem        5
#define continueItem    6
#define optionsItem     8

#define vCurrentItem    1
#define vAnalysisItem   2

#define resumeItem      1
#define fileOnlyItem    2
#define nextFileItem    3

#define appendItem      1
#define solverItem      2

/* DEFINITIONS FOR ERROR MESSAGES */

#define D_BADINPUT      20001L
#define D_PNOROOM       20002L
#define D_MISMATCH      20003L
#define D_BADMODEL      20101L
#define D_NOCONVERGENCE
    20501L

/* DEFINITIONS FOR STATUS FLAGS */

#define isInited        1
#define runPosAn        2
#define runLPos         4
#define runLDev         8
#define isListProcess   16
#define isInterrupted   32
#define appOnAbort      64
#define viewOldPos      128
#define messagesFull    256
```

```
#define  abortFlag        16384
#define quitFlag          32768
#define everyFlag         65535
```

C.3 MechMain.c

```
/*   Main Program file for linkage analysis program, */
/*   by Donald Kelly (January 1990) */

#include<NuToolsNumericalP.h>
#define _THIS_IS_MAIN_
#include "linkage.h"

FILE          *gTheInFile = NULL;
int           nvar = 0;      /* number of variables in the parameter matrix */
int           ndep = 6;
    /* number of dependent variables */
int           ntran = 0;
    /* number of transformations */
int           pindep = 0;
    /* position of independent variable in parameter matrix */

/*   GLOBAL VARIABLES FOR MENUS, EVENTS GO HERE
    */


MenuHandle      gMyMenus[7];

WindowPtr       gMyWindow;
int             gMaxRow, gCurRow;
Rect          gDragRect;

EventRecord     gTheEvent;
Boolean         gWNEImplemented;
matrix          gVar, gPositionAnalysis;
data          *gTheLinkage;
long          gErrorFlag;
unsigned int  gStatusFlag = 64;

int             lastWritten = 0;
int             nextSent = 0;
char          *messageList[15];


main()
{
    long    error;
```

```
    INIT_MAC;

    SysBeep(12);
    BuildMenus();
    SysBeep(12);
    WindowInit();
    IsWNEImplemented();
    gVar = InitM(&error);
    gPositionAnalysis = InitM(&error);
    InitSolver();
    SayFirstMessage();

    while(!(gStatusFlag & quitFlag)) {
        Loop();
    }

    bye();
}

void BuildMenus()
{
    Handle      myMenuBar;
    int         i;

    myMenuBar = GetNewMBar(BASE_RES_ID);
    SetMenuBar(myMenuBar);
    for(i = 0 ; i <= viewM ; ++i)
        gMyMenus[i] = GetMHandle(BASE_RES_ID + i);

    AddResMenu(gMyMenus[appleM], 'DRVR');

    gMyMenus[continueM] = GetMenu(continueID);
    gMyMenus[optionsM] = GetMenu(optionsID);
    InsertMenu(gMyMenus[continueM], -1);
    InsertMenu(gMyMenus[optionsM], -1);
    CheckItem(gMyMenus[optionsM], appendItem, TRUE);

    DrawMenuBar();

}

void WindowInit()
{
    Rect    theWindowRect;
    int         i;

    gMyWindow = GetNewWindow(BASE_RES_ID, 0L, -1L);
    SetPort(gMyWindow);

    theWindowRect = gMyWindow->portRect;
```

```
    gMaxRow = theWindowRect.bottom - theWindowRect.top - 15;
    gCurRow = 0;

    gDragRect = screenBits.bounds;
    gDragRect.left += 30;
    gDragRect.right -= 30;
    gDragRect.bottom -= 30;

    TextFont(monaco);
    TextSize(12);

    ShowWindow(gMyWindow);
    SelectWindow(gMyWindow);

    for(i = 0 ; i < 15 ; ++i)
        messageList[i] = (char*) malloc(256L);
}

void IsWNEImplemented()
{
    gWNEImplemented = (NGetTrapAddress(0x60, ToolTrap) != NGetTrapAd-
    dress(0x9F, ToolTrap));
}

void Loop()
{
    int             wpart;
    WindowPtr   wptr;
    Boolean yesEvent;


    if(gWNEImplemented)
        yesEvent = WaitNextEvent(everyEvent, &gTheEvent, 0L, 0L);
    else {
        SystemTask();
        yesEvent = GetNextEvent(everyEvent, &gTheEvent);
    }

    if(yesEvent) {
        switch(gTheEvent.what) {
            case mouseDown :
                if(!(gStatusFlag & (runPosAn + runLPos + runLDev)) ||
                (gStatusFlag & isInterrupted)) {
                    wpart = FindWindow(gTheEvent.where, &wptr);
                    switch(wpart) {
                        case inMenuBar :

    AdjustMenus();

    do_menu(MenuSelect(gTheEvent.where));
```

```
        break;
                                case inSysWindow :

        SystemClick(&gTheEvent, wptr);

        break;
                                case inContent :

        SelectWindow(wptr);

        break;
                                case inDrag :

        DragWindow(wptr, gTheEvent.where, &gDragRect);

        break;
                                default :

        break;
                            }
                        }
                        break;
                    case keyDown :
                        if(gTheEvent.modifiers & cmdKey) {
                            AdjustMenus();
                            do_menu(MenuKey(gTheEvent.message & charCodeMask));
                        }
                        break;
                    case activateEvt :
                        break;
                    case updateEvt :
                        if((WindowPtr) gTheEvent.message == gMyWindow) {
                            BeginUpdate(gTheEvent.message);
                            UpdateScreen(TRUE);
                            EndUpdate(gTheEvent.message);
                        }
                        break;
                    default :
                        break;
                }
        }

}

void do_menu(mResult)
        long            mResult;
{
        register int    theItem, theMenu;
        int             i;
        Str255          name;
```

```
    if(mResult != 0L) {
        theItem = LoWord(mResult);
        theMenu = HiWord(mResult);

        switch(theMenu) {
            case appleID :
                if(theItem == aboutItem) {
                    NoteAlert(BASE_RES_ID, 0L);
                } else {
                    GetItem(gMyMenus[appleM], theItem, &name);
                    OpenDeskAcc(&name);
                }
                break;
            case fileID :
                switch(theItem) {
                    case openItem :
                        GetNewLinkage(&gVar, &gTheLinkage, &gErrorFlag);
                        SelectWindow(gMyWindow);
                        break;
                    case closeItem :
                        CloseLinkage();
                        break;
                    case quitItem :
                        gStatusFlag |= quitFlag;
                        gStatusFlag -= (gStatusFlag & isInterrupted);
                        break;
                    default :
                        break;
                }
                break;
            case editID :
                if(!SystemEdit(theItem-1)) SysBeep(2);
                break;
            case viewID :
                if(theItem == vCurrentItem)
                    ViewCurrentPosition(&gVar);
                else SysBeep(16);
                break;
            case runID :
            case continueID :
            case optionsID :
                DoRunMenu(mResult);
                break;
            default :
                SysBeep(16);
                break;
        }
    }
    HiliteMenu(0);
    DrawMenuBar();
}
```

```
Boolean IsDAWindow(whichWindow)
    WindowPtr       whichWindow;
{
    if(whichWindow == 0L)
        return(FALSE);
    else
        return(((WindowPeek)whichWindow)->windowKind < 0);
}

void AdjustMenus()
{
    if(IsDAWindow(FrontWindow())) {
        EnableItem(gMyMenus[editM], undoItem);
        EnableItem(gMyMenus[editM], cutItem);
        EnableItem(gMyMenus[editM], copyItem);
        EnableItem(gMyMenus[editM], pasteItem);
        EnableItem(gMyMenus[editM], clearItem);
    } else {
        DisableItem(gMyMenus[editM], undoItem);
        DisableItem(gMyMenus[editM], cutItem);
        DisableItem(gMyMenus[editM], copyItem);
        DisableItem(gMyMenus[editM], pasteItem);
        DisableItem(gMyMenus[editM], clearItem);
    }
}

void bye()
{
    long error;
    int i;

    DeallocateM(&gVar, &error);
    DeallocateM(&gPositionAnalysis, &error);
    free(gTheLinkage);
    if(gTheInFile)
        CloseLinkage();
    for(i = 0 ; i < 15 ; ++i)
        free(messageList[i]);
}
```

C.4 MechSolve.c

```
/*  file containing the analysis routines, and their control */
/*  routines, for the Mechanism Program. This, along with */
/*  position_routines.c, is the heart of this program - all the */
```

```
/*   rest is just a shell for easier use. Don Kelly, Jan 1990 */

#include <NuToolsNumericalF.h>
#include "linkage.h"

static matrix          oldPosition;
static int             numSteps = 2;
static long                  theOutVar = 5L;
static double          runLength = 1.5707961;
static double          runIncrement = 0.1745329;
static double          theSpan, dwellPosition;
FILE                         *theOutFile = NULL;

void InitSolver()
{
    if(gStatusFlag & isInited) return;
    oldPosition = InitM(&gErrorFlag);
    gStatusFlag |= isInited;
}

void RunSolution()
{
    double      stopVal, stepSize;
    int         numPassed, i;

    EnableItem(gMyMenus[runM], stopItem);
    DisableItem(gMyMenus[viewM], 0);
    DisableItem(gMyMenus[runM], posItem);
    DisableItem(gMyMenus[runM], listPosItem);
    DisableItem(gMyMenus[runM], listDevItem);
    DisableItem(gMyMenus[runM], optionsItem);
    DisableItem(gMyMenus[fileM], openItem);
    DisableItem(gMyMenus[fileM], closeItem);
    WriteMessage("In RunSolution()\0");
    HiliteMenu(0);
    DrawMenuBar();
    SelectWindow(gMyWindow);

    stopVal = gVar.mat[pindep][0] + runLength;
    stepSize = runIncrement/numSteps;
    while( !(gStatusFlag & (quitFlag + abortFlag)) && !gErrorFlag) {

        do {
            FindPosition();
        } while(gStatusFlag & isInterrupted);
        if(!(gStatusFlag & (quitFlag + abortFlag)))
            TempWrite();

        if(gVar.mat[pindep][0] >= stopVal) {
            gStatusFlag -= gStatusFlag & (runPosAn + runLPos + runLDev);
            gStatusFlag |= abortFlag;
```

```
        } else {
            if(!(gStatusFlag & (abortFlag + quitFlag)))
                gVar.mat[pindep][0] += stepSize;
        }

        for(i = 1 ; i < numSteps && !gErrorFlag && !(gStatusFlag & (quitFlag +
        abortFlag)) ; ++i, gVar.mat[pindep][0] += stepSize) {
            FindPosition();
            while(gStatusFlag & isInterrupted)
                Loop();
        }

        if((gStatusFlag & (abortFlag + quitFlag)) && (gStatusFlag & (runPosAn +
        runLPos+ runLDev)) && (gStatusFlag & appOnAbort)) {
            WriteMessage("Data from unfinished run:\0");
            TempWrite();
        }

    }



    gStatusFlag -= (gStatusFlag & abortFlag);
    EnableItem(gMyMenus[fileM], closeItem);
    EnableItem(gMyMenus[runM], posItem);
    EnableItem(gMyMenus[runM], listPosItem);
    EnableItem(gMyMenus[runM], listDevItem);
    EnableItem(gMyMenus[runM], optionsItem);
    EnableItem(gMyMenus[viewM], 0);
    DisableItem(gMyMenus[runM], stopItem);
    DisableItem(gMyMenus[runM], continueItem);
    WriteMessage("Exiting RunSolution\0");
    gStatusFlag -= gStatusFlag & (runPosAn + runLPos + runLDev);
    /* CODE NOT DONE YET! */
}

void DoRunMenu(mResult)
    long    mResult;
{
    int             theMenu, theItem;

    theMenu = HiWord(mResult);
    theItem = LoWord(mResult);

    switch(theMenu) {
        case runID:
            switch(theItem) {
                case posItem :
                    gStatusFlag |= runPosAn;
                    DisableItem(gMyMenus[continueM], fileOnlyItem);
                    DisableItem(gMyMenus[continueM], nextFileItem);
```

```
                RunSolution();
                break;
            case listPosItem :
                gStatusFlag |= runLPos;
                EnableItem(gMyMenus[continueM], fileOnlyItem);
                EnableItem(gMyMenus[continueM], nextFileItem);
                ListProcess();
                break;
            case listDevItem :
                gStatusFlag |= runLDev;
                EnableItem(gMyMenus[continueM], fileOnlyItem);
                EnableItem(gMyMenus[continueM], nextFileItem);
                ListProcess();
                break;
            case stopItem :
                if(gStatusFlag & isInterrupted) {
                    gStatusFlag |= abortFlag;
                    gStatusFlag -= (gStatusFlag & isListProcess);
                    SetItem(gMyMenus[runM], stopItem, "\pStop");
                } else {
                    EnableItem(gMyMenus[runM], continueItem);
                    EnableItem(gMyMenus[viewM], 0);
                    SetItem(gMyMenus[runM], stopItem, "\pAbort");
                    WriteMessage("Solution run interrupted\0");
                }
                gStatusFlag ^= isInterrupted;
                EnableItem(gMyMenus[runM], optionsItem);
                break;
            default :
                break;
        }
        break;
    case continueID :
        DisableItem(gMyMenus[runM], optionsItem);
        DisableItem(gMyMenus[runM], continueItem);
        DisableItem(gMyMenus[viewM], 0);
        SetItem(gMyMenus[runM], stopItem, "\pStop");
        gStatusFlag ^= isInterrupted;
        switch(theItem) {
            case fileOnlyItem :
                gStatusFlag -= (gStatusFlag & isListProcess);
                break;
            case nextFileItem :
                gStatusFlag |= abortFlag;
                break;
            default :
                break;
        }
        SelectWindow(gMyWindow);
        WriteMessage("Solution resumed\0");
        break;
```

```
            case optionsID :
                switch(theItem) {
                    case appendItem :
                        if(gStatusFlag & appOnAbort)
                            CheckItem(gMyMenus[optionsM], appendItem, FALSE);
                        else
                            CheckItem(gMyMenus[optionsM], appendItem, TRUE);
                        gStatusFlag ^= appOnAbort;
                        break;
                    case solverItem :
                        SetSolverControl(&numSteps, &runIncrement, &runLength);
                        break;
                    default :
                        break;
                }
                break;
            default :
                break;
        }

}

void FindPosition()
{
    int             i;

    Loop();

    for(i = 0 ; i < 15 && !gErrorFlag && !(gStatusFlag & (abortFlag + quitFlag +
    isInterrupted)) ; ++i) {
        if(iterate(&gVar, gTheLinkage, &gErrorFlag) <= 1e-6) break;
        Loop();
    }
    if(i == 15) gErrorFlag = D_NOCONVERGENCE;

}

void TempWrite()
{
    int             numPassed;
    char    *theString;

    if(gStatusFlag & isListProcess) {
        if(gStatusFlag & runLDev)
            WriteOutput(theOutFile, gVar.mat[pindep][0],
            fabs(gVar.mat[theOutVar][0] - dwellPosition)/theSpan, &gErrorFlag);
        else
            WriteOutput(theOutFile, gVar.mat[pindep][0], gVar.mat[theOutVar][0],
            &gErrorFlag);
    }
```

```
    theString = (char *) malloc(256L);
    if(!theString)
        WriteMessage("Sorry, can't write data!\0");
    else {
        numPassed = sprintf(theString, "%10.6g    %10.6g", gVar.mat[pindep][0],
        gVar.mat[theOutVar][0]);
        WriteMessage(theString);
        free(theString);
    }
}


void ListProcess()
{
    char    *outFileName;
    int         whatsRunning, junk;

    outFileName = (char *) malloc(100L);
    if(!outFileName) return;

    gStatusFlag |= isListProcess;
    whatsRunning = gStatusFlag & (runLPos + runLDev);

    while(!(gStatusFlag & quitFlag) && (gStatusFlag & isListProcess) &&
    !gErrorFlag && ReadNextTrial(&gVar, outFileName, &theSpan, &theOutVar,
    &gErrorFlag)) {
        gStatusFlag |= whatsRunning;
        theOutFile = fopen(outFileName, "a");
        if(!theOutFile)
            gStatusFlag ^= isListProcess;
        else {
            dwellPosition = gVar.mat[theOutVar][0];
            gStatusFlag |= whatsRunning;
            RunSolution();
            junk = fclose(theOutFile);
        }
    }
    free(outFileName);
    gStatusFlag -= (gStatusFlag & whatsRunning);
    /*  CODE NOWHERE NEAR DONE YET! */

}
```

C.5 position_routines.c

```
/*  routines to perform tasks involved with position analysis of
    spatial mechanisms, using the methods described in ref. [16]
    of my thesis - by Don Kelly, Sept 1989 */
```

```
#include <NuToolsNumericalF.h>
#include "linkage.h"

void setup(var, p, error)
    matrix      *var;
    data    **p;
    long    *error;
{
    int         i, j, k, num_passed;
    int         n, x, y, z, theta, lead;
    float   a[5];
    data    *pn;

    if(*error) return;

    num_passed = printf("How many transformations, parameters ? ");
    if(scanf("%d %d", &ntran, &k) != 2) {
        *error = D_BADINPUT;
        return;
    }
    num_passed = printf("How many variables, dependent variables ? ");
    if(scanf("%d %d", &nvar, &ndep) != 2) {
        *error = D_BADINPUT;
        return;
    }
    num_passed = printf("Enter position of independent variable: ");
    if(scanf("%d", &pindep) != 1) {
        *error = D_BADINPUT;
        return;
    }

    AllocateM(var, k, 1, error);
    if(! *error) *p = (data *) malloc((size_t) ntran*sizeof(data)); else return;
    if(*p == NULL)
        *error = D_PNOROOM;
    if(*error) return;

    for(i = 0, pn = *p ; i < ntran && !(*error) ; ++i, ++pn) {
        num_passed = printf("Enter the parameters for transformation %d: ", i+1);
        if(scanf("%d %d %d %d %d %d", &n, &x, &y, &z, &theta, &lead) == 6) {
            pn->n = n;
            pn->dx = x;
            pn->dy = y;
            pn->dz = z;
            pn->ang = theta;
            pn->lead = lead;
        } else *error = D_BADINPUT;
    }

    for(i = 0; i < k && !(*error) ; i += 5) {
```

```
        num_passed = printf("Enter five original parameter values:");
        if(scanf("%f %f %f %f %f", &a[0], &a[1], &a[2], &a[3], &a[4]) == 5) {
            for(j = 0 ; j < 5 && i+j < k ; ++j)
                var->mat[i+j][0] = a[j];
        } else *error = D_BADINPUT;
    }
}

void transform(s, var, p, error)
    matrix      *s;
    matrix      *var;
    data    *p;
    long    *error;
{
    double      ct, st, theta, screw;
    matrix      t, dum;

    if(*error) return;

    t = InitM(error);
    dum = InitM(error);
    MakeIdentityM(&t, 4, error);
    CopyM(&dum, s, error);

    theta = (p->ang < 0 ? 0.0 : var->mat[p->ang][0]);
    screw = (p->lead < 0 ? 0.0 : 0.5*var->mat[p->lead][0]*theta/PI);
    ct = cos(theta);
    st = sin(theta);
    t.mat[0][3] = (p->dx < 0 ? 0.0 : var->mat[p->dx][0]);
    t.mat[1][3] = (p->dy < 0 ? 0.0 : var->mat[p->dy][0]);
    t.mat[2][3] = (p->dz < 0 ? 0.0 : var->mat[p->dz][0]);

    switch ( (p->n + 2) % 3 + 1) {
        case 1 :
            t.mat[1][1] = ct;
            t.mat[1][2] = -st;
            t.mat[2][1] = st;
            t.mat[2][2] = ct;
            t.mat[0][3] += screw;
            break;
        case 2 :
            t.mat[0][0] = ct;
            t.mat[0][2] = st;
            t.mat[2][0] = -st;
            t.mat[2][2] = ct;
            t.mat[1][3] += screw;
            break;
        case 3 :
            t.mat[0][0] = ct;
            t.mat[0][1] = -st;
            t.mat[1][0] = st;
```

```
            t.mat[1][1] = ct;
            t.mat[2][3] += screw;
            break;
    }

    MultM(s, &dum, &t, error);

    DeallocateM(&t, error);
    DeallocateM(&dum, error);

}

void get_s(k, s, var, p, error)
    matrix      *s, *var;
    data    *p;
    int         k;
    long    *error;
{
    int         i, j;
    data    *pn;

    if(*error) return;
    j = (k > ntran-1 ? ntran-1 : k);
    if(var->n != 1) *error = 1;
    MakeIdentityM(s, 4, error);

    for(i = 0, pn = p ; i <= j && !(*error) ; ++i, ++pn)
        transform(s, var, pn, error);
}

void invert_s(s, sinv, error)
    matrix      *s, *sinv;
    long    *error;
{
    int         i, j;

    if(!(*error)) {
        if(s->m != 4 || s->n != 4) *error = 1;
        else MakeIdentityM(sinv, 4, error);
    }
    if (*error) return;

    for( i = 0 ; i < 3 ; ++i) {
        sinv->mat[i][i] = s->mat[i][i];
        for(j = 0 ; j< 3 ; ++j) {
            sinv->mat[i][3] -= s->mat[i][j]*s->mat[i][3];
            if(j > i) {
                sinv->mat[i][j] = s->mat[j][i];
                sinv->mat[j][i] = s->mat[i][j];
            }
        }
```

```
    }

}

void get_r(r, var, p, error)
    long    *error;
    matrix      *r, *var;
    data    *p;
{
    matrix      s, si;

    if(*error) return;

    s = InitM(error);
    si = InitM(error);
    AllocateM(&s, 4, 4, error);
    AllocateM(&si, 4, 4, error);
    AllocateM(r, 6, 1, error);

    get_s(ntran, &s, var, p, error);
    invert_s(&s, &si, error);

    if (!(*error)) {
        r->mat[0][0] = si.mat[0][2] + si.mat[0][0] + si.mat[2][2] - 2;
        r->mat[1][0] = si.mat[1][0] + si.mat[0][0] + si.mat[1][1] - 2;
        r->mat[2][0] = si.mat[2][1] + si.mat[1][1] + si.mat[2][2] - 2;
        r->mat[3][0] = si.mat[0][3];
        r->mat[4][0] = si.mat[1][3];
        r->mat[5][0] = si.mat[2][3];
    }

    DeallocateM(&s, error);
    DeallocateM(&si, error);

}

void get_d(s, d, var, p, error)
    matrix      *s, *d, *var;
    data    *p;
    long    *error;
{
    int         i, j;
    double      screw;

    if(*error) return;
    MakeZeroM(d, 6, 1, error);

    i = (p->n + 2) % 3;

    if(p->n > 6) {
        for(j = 0 ; j < 3 ; ++j)
```

```
            d->mat[j+3][0] = s->mat[j][i];
    } else {
        d->mat[0][0] = s->mat[1][i];
        d->mat[1][0] = s->mat[2][i];
        d->mat[2][0] = s->mat[0][i];
        d->mat[3][0] = s->mat[1][3]*s->mat[2][i] - s->mat[2][3]*s->mat[1][i];
        d->mat[4][0] = s->mat[2][3]*s->mat[0][i] - s->mat[0][3]*s->mat[2][i];
        d->mat[5][0] = s->mat[0][3]*s->mat[1][i] - s->mat[1][3]*s->mat[0][i];
        if(p->n > 3) {
            screw = 0.5 * var->mat[p->lead][0]/PI;
            for(j = 0 ; j < 3 ; ++i)
                d->mat[j+3][0] += screw*s->mat[j][i];
        }
    }
}

double iterate(var, p, error)
    matrix      *var;
    data        *p;
    long        *error;
{
    matrix      s, r, delta_var;
    data        *pn;
    double      biggest;
    int         i, j;
    long        er;

    if(*error) return;

    s = InitM(error);
    r = InitM(error);
    delta_var = InitM(error);
    AllocateM(&delta_var, ndep, ndep+1, error);

    get_r(&r, var, p, error);
    for(i = 0 ; i < ndep ; ++i)
        delta_var.mat[i][ndep] = r.mat[i][0];

    for(i = 0 ; i < ndep ; ++i) {
        for(j = 0, pn = p ; j < ntran ; ++j, ++pn) {
            if(pn->dx == i) break;
            if(pn->dy == i) break;
            if(pn->dz == i) break;
            if(pn->ang == i) break;
        }
        if(j < ntran) get_s(j, &s, var, p, error); else *error = D_BADMODEL;
        get_d(&s, &r, var, pn, error);
        for(j = 0 ; j < ndep ; ++j)
            delta_var.mat[j][i] = r.mat[j][0];
    }
```

```
    GaussElimM(&r, &delta_var, 1e-10, NU_PARTIAL_PIVOT, error);
    if(!(*error)) BacksubM(&delta_var, &r, error);

    for(i = 0, biggest = 0.0 ; i < ndep && !(*error) ; ++i) {
        var->mat[i][0] += delta_var.mat[i][0];
        if(biggest < fabs(delta_var.mat[i][0])) biggest = fabs(delta_var.mat[i][0]);
    }

    DeallocateM(&s, &er);
    DeallocateM(&r, &er);
    DeallocateM(&delta_var, &er);
    return(biggest);
    /* code not done yet- you need error checking */
    /* in this and other routines */
}

void f_setup(theFile, var, p, error)
    FILE        *theFile;
    matrix      *var;
    data        **p;
    long        *error;
{
    int             i, j, k, num_passed;
    int             n, x, y, z, theta, lead;
    data        *pn;
    matrix          temp;
    long        localError;

    if(*error) return;
    localError = 0L;
    temp = InitM(error);

    if(fscanf(theFile, "%d %d", &ntran, &k) != 2) {
        *error = D_BADINPUT;
        return;
    };
    if(fscanf(theFile, "%d %d", &nvar, &ndep) != 2) {
        *error = D_BADINPUT;
        return;
    };
    if(fscanf(theFile, "%d", &pindep) != 1) {
        *error = D_BADINPUT;
        return;
    };

    AllocateM(var, k, 1, error);
    if(! *error) *p = (data *) malloc((size_t) ntran*sizeof(data)); else return;
    if(*p == NULL) {
        *error = D_PNOROOM;
        return;
    }
```

```
    for(i = 0, pn = *p ; i < ntran && !(*error) ; ++i, ++pn) {
        if(fscanf(theFile, "%d %d %d %d %d %d", &n, &x, &y, &z, &theta, &lead)
        == 6) {
            pn->n = n;
            pn->dx = x;
            pn->dy = y;
            pn->dz = z;
            pn->ang = theta;
            pn->lead = lead;
        } else *error = D_BADINPUT;
    }

    if(*error) return;

    AllocateM(&temp, 1, k, error);
    if(!(*error))
        fReadM(&temp, theFile, error);
    if(!(*error)) {
        if(temp.m == var->n && temp.n == var->m)
            TransposeM(var, &temp, error);
        else *error = D_MISMATCH;
    }

    DeallocateM(&temp, &localError);
}
```

C.6 MechFileControl.c

```
/*  file containing file control subroutines for the main */
/*  linkage program, by Donald Kelly (January 1990) */

#include <NuToolsNumericalF.h>
#include "linkage.h"

void GetNewLinkage(var, p, error)
    matrix      *var;
    data    **p;
    long    *error;
{
    int         junk;
    GrafPtr     oldPort;

    gTheInFile = OpenInputFile("r", error);

    if(!*error) {
        f_setup(gTheInFile, var, p, error);
```

```
        if(*error) {
            ErrorMessage(*error, "That file could not be opened!", 2, stopIcon);
            junk = fclose(gTheInFile);
            gTheInFile = NULL;
            return;
        }

        /* CODE TO GO HERE FOR ENABLING MENU ITEMS */
        EnableItem(gMyMenus[viewM], 0);
        EnableItem(gMyMenus[runM], 0);
        EnableItem(gMyMenus[fileM], closeItem);
        DisableItem(gMyMenus[fileM], openItem);
    }
    GetPort(&oldPort);
    SetPort(gMyWindow);
    InvalRect(&gMyWindow->portRect);
    SetPort(oldPort);
}

void CloseLinkage()
{
    int junk;

    junk = fclose(gTheInFile);
    gTheInFile = NULL;
    DisableItem(gMyMenus[viewM], 0);
    DisableItem(gMyMenus[runM], 0);
    DisableItem(gMyMenus[fileM], closeItem);
    EnableItem(gMyMenus[fileM], openItem);

}

Boolean ReadNextTrial(var, outFileName, span, outVar, error)
    matrix      *var;
    char        *outFileName;
    double      *span;
    long        *outVar, *error;
{
    char        *theMessage, *otherMessage;
    matrix      temp;
    int         numPassed;

    error = 0L;
    temp = InitM(error);
    AllocateM(&temp, 5, 1, error);

    theMessage = (char*) malloc(100L);
    if(!theMessage || !gTheInFile) {
        *error = 1L;
        return(FALSE);
    }
```

```
    MoveToNextLine(gTheInFile);
    otherMessage = fgets(theMessage, 100, gTheInFile);
    if(feof(gTheInFile)) {
        *error = EOF;
        return(FALSE);
    }
    if(sscanf(theMessage, "%s %lg %ld", outFileName, span, outVar) != 3) {
        *error = 2L;
        return(FALSE);
    }

    fReadM(&temp, gTheInFile, error);
    if(!(*error) && temp.m == var->n && temp.n == var->m) {
        TransposeM(var, &temp, error);
        DeallocateM(&temp, error);
    } else {
        if(!(*error)) *error = 3L;
    }

    if(*error) return(FALSE);
}

void MoveToNextLine(theFile)
    FILE        *theFile;
{
    Boolean     inLoop;
    int         theChar;

    inLoop = TRUE;

    while(inLoop && !feof(theFile)) {
        theChar = fgetc(theFile);
        if(!isspace(theChar)) {
            inLoop = FALSE;
            theChar = ungetc(theChar, theFile);
        }
    }
}

void DumpPositionAnalysis(outFile, pos, error)
    FILE        *outFile;
    matrix      *pos;
    long    *error;
{
    long        row, col;
    int         numPassed;
    char    theChar;

    numPassed = fprintf(outFile, "%ld\t%ld\n", pos->n, pos->m);
```

```
    for(row = 0 ; row < pos->m ; ++row) {
        for(col = 0 ; col < pos->n ; ++col) {
            if(col == pos->n - 1)
                theChar = '\n';
            else
                theChar = '\t';
            numPassed = fprintf(outFile, "%lg%c", pos->mat[row][col], theChar);
        }
    }
}

void WriteOutput(theOutFile, firstVar, secondVar, error)
    FILE        *theOutFile;
    double      firstVar, secondVar;
    long    *error;
{
    int numPassed;

    numPassed = fprintf(theOutFile, "%lg\t%lg\n", firstVar, secondVar);
}
```

C.7 MechView.c

```
/*  file to hold view commands and other simple chores */
/*  for my Mechanism program, by Don Kelly (January 1990) */

#include <NuToolsNumericalF.h>
#include "linkage.h"


void ViewCurrentPosition(thePos)
    matrix *thePos;
{
    WriteM(thePos, 10, 12, "Current Position", &gErrorFlag);
}

void ViewPositionAnalysis()
{
    WriteM(&gPositionAnalysis, 10, 12, "PositionAnalysis", &gErrorFlag);
}

void ScrollMyWindow()
{
    RgnHandle       tempRgn;

    tempRgn = NewRgn();
    ScrollRect(&gMyWindow->portRect, 0, -15, tempRgn);
    DisposeRgn(tempRgn);
```

```
}

void DrawMyMessage(myMessage)
    char        *myMessage;
{
    GrafPtr     oldPort;

    GetPort(&oldPort);
    SetPort(gMyWindow);
    if(gCurRow = gMaxRow)
        ScrollMyWindow();
    else
        gCurRow += 15;

    MoveTo(5, gCurRow);
    DrawString(myMessage);
    SetPort(oldPort);
}

void SayFirstMessage()
{
    WriteMessage("This is the start of the Linkage Program, by me, Donald\0");
    WriteMessage("Kelly, as part of my requirements for the Degree of Master\0");
    WriteMessage("of Science in Mechanical Engineering.  This program can\0");
    WriteMessage("determine the values of the joint variables in single-loop\0");
    WriteMessage("spatial linkages, and uses this to perform other tasks\0");
    WriteMessage("needed for my Thesis.- DFK January 1990\0");
}

void SetSolverControl(steps, increment, runLength)
    int         *steps;
    double      *increment, *runLength;
{
    long    error;
    int         junk;
    short       sigfigs;
    DOUBLE          object[3];

    error = 0L;
    sigfigs = 6;

    object[0] = (double) *steps;
    object[1] = *increment;
    object[2] = *runLength;
    GeneralEntry(object, 3, sigfigs, NU_REAL, 0, TRUE, "Solver Control",
    &error);

    if(!error) {
        junk = (int) floor(object[0]);
        if(junk > 0 && junk <= 10)
            *steps = junk;
```

```
        if(object[1] >= 0.0174 && object[1] <= 0.175)
            *increment = object[1];
        if(object[2] > 0.0 && object[2] <= 8.0)
            *runLength = object[2];
    }
}

void UpdateScreen(updateFlag)
    Boolean updateFlag;
{
    int         i, start, stop, mPosition;
    GrafPtr     oldPort;


    GetPort(&oldPort);
    SetPort(gMyWindow);

    if(updateFlag) {
        EraseRect(&gMyWindow->portRect);
        if(gStatusFlag & messagesFull) {
            for(i = 0 , mPosition = (lastWritten+1) % 15 ; i < 15 ; ++i, mPosition =
            (mPosition+1) % 15)
                DrawMyMessage(messageList[mPosition]);
            nextSent = mPosition;
        } else {
            for(i = 0 ; i <= lastWritten ; ++i)
                DrawMyMessage(messageList[i]);
            nextSent = i % 15;
        }
    } else {
        while(nextSent != (lastWritten+1) % 15) {
            DrawMyMessage(messageList[nextSent]);
            nextSent = (nextSent+1) % 15;
        }
    }

    SetPort(oldPort);
}

void WriteMessage(messageString)
    char    *messageString;
{

    ++lastWritten;
    if(lastWritten >= 15) {
        lastWritten -= 15;
        gStatusFlag |= messagesFull;
    }

    messageList[lastWritten] = strcpy(messageList[lastWritten], messageString);
    messageList[lastWritten] = CtoPstr(messageList[lastWritten]);
```

```
    UpdateScreen(FALSE);
}
```