An Iterative Procedure for Planning Manipulator Movements

by

Michael J. Procca

A thesis
presented to the University of Manitoba
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Electrical Engineering

Winnipeg, Manitoba

ISBN 0-315-47884-5

AN ITERATIVE PROCEDURE FOR PLANNING MANIPULATOR MOVEMENTS

BY

MICHAEL J. PROCCA

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Michael J. Procca

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Michael J. Procca

ABSTRACT

An iterative procedure for planning manipulator movements in the presence of obstacles is proposed. An algorithm for the detection of geometrical intersections in three dimensions as well as a novel approach to sampling were conceived and incorporated into this procedure. A computer program simulating a hypothetical manipulator was written to demonstrate the efficacy of the procedure in solving for complicated motions within reasonable time limits.

ACKNOWLEDGEMENT

I am indebted to Dr. Steve Onyshko for suggesting the research area and for his constant guidance, advice and criticisms which have shaped this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

The purpose of this thesis is to devise an iterative procedure for planning manipulator movements and to verify the efficacy of such a procedure by a computer simulation.

1.1 Problem

The problem of planning manipulator movements in the presence of obstacles is referred to as the find-path problem. The find-path problem may be defined as:

> Given start and goal configurations of an n-degree of freedom manipulator, neither of which are in collision with any obstacle, find a sequence of configurations such that there be no collisions between the manipulator and the obstacles. In addition, the change in all n degrees of freedom from one configuration to the next should be within reasonable limits. These limits depend on the changes in the degrees of freedom with respect to one another.

The above sequence of configurations constitute the path sought, which is classified as a safe path.

1.2 Current Solutions

The solution of Lozano-Perez [6,7] employs a configuration space approach. In this approach all obstacles are transformed into configuration space obstacles. A configuration space obstacle corresponds to all configurations of the manipulator which collide with the obstacle. This may be visualized for the very simple two dimensional problem of figure 1.1. This figure shows a moveable polygon A, with fixed orientation, and an obstacle B.

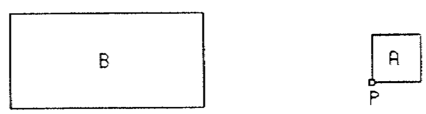


Figure 1.1 Physical space representation

If all possible locations of point P of polygon A are considered such that polygons A and B intersect, then A and B are represented in configuration space as the point P' and the configuration space obstacle B', respectively. This is shown in figure 1.2.

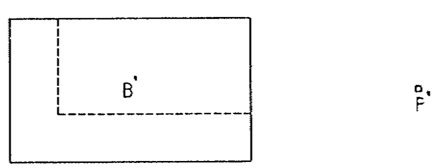


Figure 1.2 Configuration space representation

Any trajectory in configuration space which does not intersect the configuration space obstacle B' corresponds to a safe path for polygon A in physical space with respect to obstacle B.

For a three dimensional problem the configuration space obstacle would be three dimensional as well if the orientation of the moveable polyhedron is fixed, otherwise an additional dimension would have to be added for each rotation allowed. For a manipulator consisting of m different polyhedrons, m different configuration space obstacles would be required for each obstacle.

Once the configuration space obstacles have been found, all configuration space not occupied by such obstacles is broken up into convex polyhedral cells. A free space graph is constructed from these cells and searched for a connected set of cells joining cells which contain the start and goal configurations.

The solution by Gouzenes [5] builds a graph consisting of subsets of empty space, where empty space is a subset of configuration space that does not contain any part of a configuration space obstacle. This graph is then searched for a safe path using a standard graph-search technique.

The solution by Brooks [3] represents free space as a set of possibly overlapping cones. A graph is built from these cones and searched for an ordered set of nodes between the start and goal nodes.

The two solutions by Sawatzky [10] employ a collision detector which consists of bounding polyhedrons by spheres and determining whether or not the spheres intersect. If the spheres do not intersect then the two polyhedrons cannot possibly collide, otherwise more spheres would have to be used to represent the polyhedrons.

The first solution, namely generate and test, appears at the outset to be the same as an iterative approach. In this solution a path is postulated and each configuration, beginning with the start configuration, is tested with the collision detector. If a collision is detected, then the manipulator's payload is raised until it clears the obstacle and another path is postulated from the present configuration to the goal configuration. Thus this solution is still a sequential search of configurations and not a sequential search of paths.

The second solution employs the collision detector to generate a representation of free space and then uses a graph-search technique to find a safe path.

1.3 Scope

In Chapter 2 an algorithm for the detection of geometrical intersections in three dimensions is presented. The algorithm is applied to two convex polyhedrons and consists of seven steps. During any of the first six steps the relative positioning of the two polyhedrons may be determined to be safe or unsafe. If the seventh step is completed, then the relative positioning of the two polyhedrons may be determined to be safe or it may be indeterminant. For the latter case the relative positioning of the two polyhedrons must be taken as unsafe. Figure 1.3 consists of a flow chart for this algorithm.

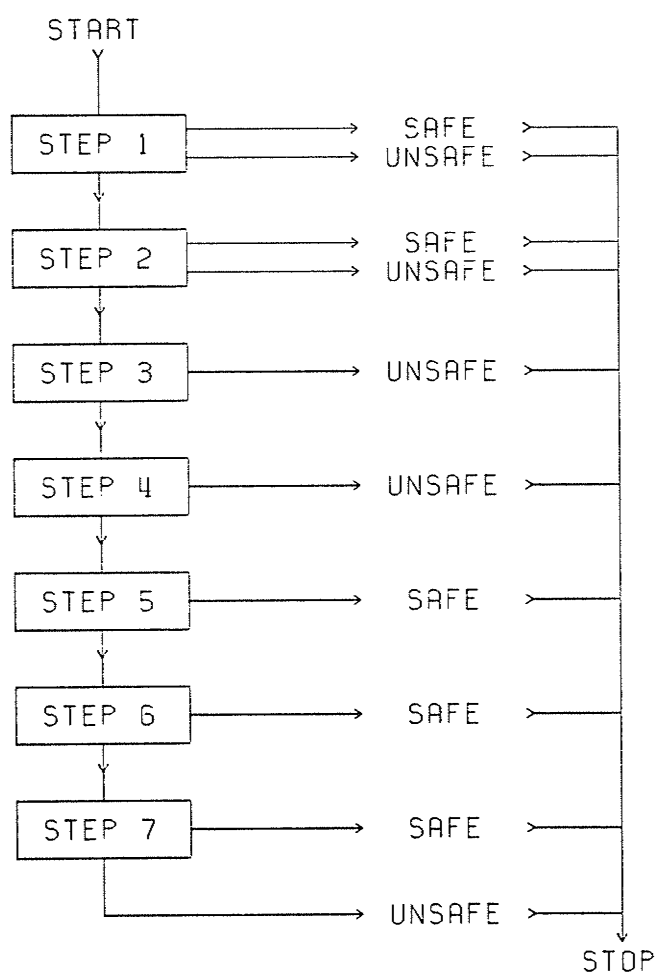


Figure 1.3 Flow chart for the algorithm of Chapter 2

In Chapter 3 a description is given of the data structures required for all convex polyhedrons which make up the manipulator, obstacles and workpieces.

In Chapter 4 a technique for selecting sample configurations, which is dependent on the location of obstacles, is described along with the iterative procedure. The iterative procedure starts with a linear path, that is one in which all the degrees of freedom vary linearly from the start to goal configurations, and determines whether or not all sample configurations are safe. Whether or not a sample configuration is safe can be ascertained by applying the algorithm of Chapter 2 to all polyhedron pairs, formed from polyhedrons of the manipulator and polyhedrons representing the obstacles and workpieces. If not all configurations are safe, then the unsafe configurations are modified and some or all of the safe configurations may be modified to preserve continuity of the path. The aforementioned procedure is repeated until a safe path is found or until the maximum number of iterations is reached. Figure 1.4 consists of a flow chart for the iterative procedure.

A Pascal computer program, namely FINDPATH in Appendix A, was written to illustrate the efficacy of the previously mentioned ideas in solving the find-path problem. The output of this program is contained in Appendix B.

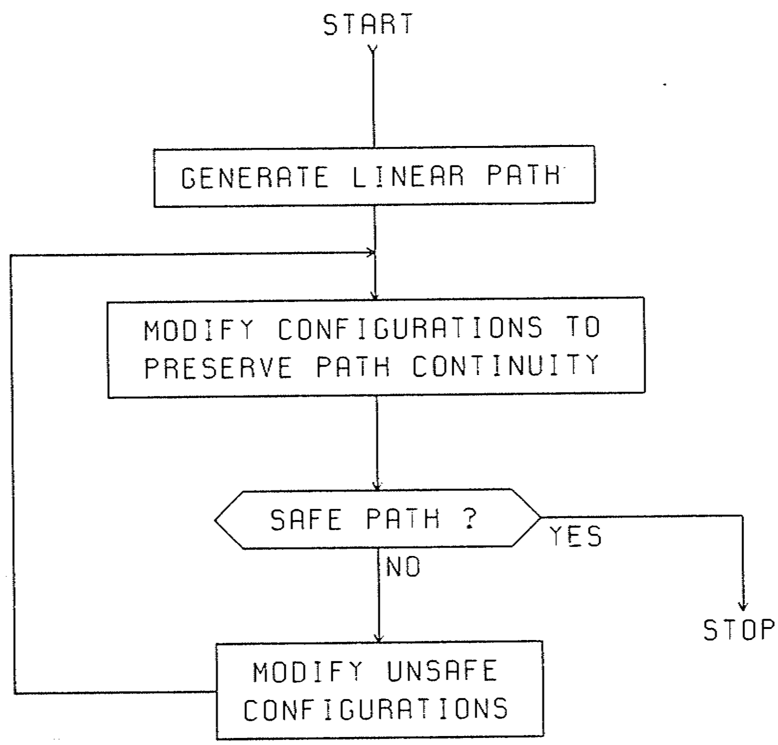


Figure 1.4 Flow chart for the iterative procedure

This thesis is concerned purely with the geometrical aspects of the find-path problem and not with the dynamics required of the manipulator to execute any path found. The manipulator simulated by the program FINDPATH is a hypothetical one. Its purpose is merely to facilitate demonstration of the intersection detection algorithm, the sample selection technique and the iterative procedure.

CHAPTER 2

An Algorithm for Detection of Geometrical Intersections in Three Dimensions

In this chapter an algorithm is presented for determining whether or not two convex polyhedrons intersect. The property of convexity, namely that for any plane face the polyhedron lies entirely on one side of the infinite plane containing the plane face, is of paramount importance since the algorithm is based on this property.

2.1 Prerequisites

The algorithm requires the following representation of, and information on, each polyhedron:

I) Each vertex is assigned an integer referred to as **VERTEXNUMBER** which is unique amoung the **VERTEXNUMBER**s of the polyhedron.

II) The set of all **VERTEXNUMBER**s of the polyhedron is referred to as **POLYVERTEXSET**.

III) Each plane face is assigned an integer referred to as **PLANENUMBER** which is unique amoung the **PLANENUMBER**s of the polyhedron.

IV) The set of all **PLANENUMBERs** of the polyhedron is referred to as **POLYPLANESET.**

V) For each plane face the set of all **VERTEXNUMBERs** of vertices which lie in the plane face is referred to as **PLANEVERTEXSET.**

VI) For each vertex the set of all **PLANENUMBERs** of plane faces that the vertex lies in is referred to as **VERTEXPLANESET.**

VII) The coordinates of each vertex with respect to a global cartesian coordinate system.

VIII) The outward unit normal vector to each plane face represented in the same global Cartesian coordinate system mentioned in **VII.**

Requirements **I** to **VI** are functions of the particular polyhedron itself, while requirements **VII** and **VIII** are dependant on the polyhedron's location and orientation.

In addition, the algorithm requires the definition of displacement of a vertex with a plane face as the signed, minimum distance of the vertex from the infinite plane containing the plane face. Positive (Negative) indicates that the vertex lies on the opposite (same) side of the infinite plane from (as) that of the polyhedron which the plane face belongs to. To calculate the displacement of a vertex with a plane face, simply take the inner product of a vector, formed by the vertex and a vertex belonging to the plane face and pointing to the vertex, and the plane face's outward unit normal vector.

2.2 Notation

The symbols employed for the sets and set operators are the same as those employed by the Pascal language to facilitate following the implementation of the algorithm in procedure CHECKSAFENESS from page 75 to 83 in Appendix A. Sets are represented either by name or by their elements enclosed in square brackets. The empty set is represented by []. The set operators union, intersection, difference, equality and inequality are represented by +, *, -, = and <>, respectively. In addition, the word **IN** is used to represent an operator which determines whether of not an element belongs to a set and returns a Boolean result of TRUE or FALSE, respectively.

2.3 Algorithm

Consider any two convex polyhedrons A and B. Polyhedrons A and B are composed of vertices with **VERTEXNUMBERs** i=1,2,...k and i=1,2,...l, respectively, and plane faces with **PLANENUMBERs** j=1,2,...m and j=1,2,...n, respectively.

2.3.1 Step 1

For a plane face of polyhedron A with **PLANENUMBER**=j calculate displacements with all vertices of polyhedron B and form a set of **VERTEXNUMBER**s corresponding to all positive displacement vertices. Refer to this set as **PDVERTEXSET**(A,j). If:

PDVERTEXSET(A,j) = **POLYVERTEXSET**(B) (1)

then polyhedrons A and B cannot intersect since all vertices of polyhedron B are on the opposite side of an infinite plane, which contains the plane face of polyhedron A with **PLANENUMBER**=j, than that of polyhedron A. Therefore, the relative positioning of polyhedrons A and B is safe.

For a vertex of polyhedron B with **VERTEXNUMBER**=i calculate displacements with all plane faces of polyhedron A and form a set of **PLANENUMBER**s corresponding to all negative displacement plane faces. Refer to this set as **NDPLANESET**(B,i). If:

NDPLANESET(B,i) = **POLYPLANESET**(A) (2)

then polyhedrons A and B intersect since all plane faces of polyhedron A have a vertex of polyhedron B, with **VERTEXNUMBER**=i, on the same side of their corresponding infinite planes as that of polyhedron A itself. In other words, this vertex of polyhedron B is inside of polyhedron

A. Therefore, the relative positioning of polyhedrons A and B is unsafe.

If neither j, j=1,2,...m, nor i, i=1,2,...l, can be found such that either (1) or (2) is true, then the relative positioning of polyhedrons A and B is indeterminant at this point.

2.3.2 Step 2

If the relative positioning of polyhedrons A and B is still indeterminant, then repeat step 1 but exchange the roles of polyhedrons A and B.

2.3.3 Step 3

If the relative positioning of polyhedrons A and B is still indeterminant, then if an intersection occurs it will consist of one or more edges of one polyhedron piercing the other since no vertex of either polyhedron lies inside the other.

For a vertex of polyhedron A with **VERTEXNUMBER**=i consider all the plane faces corresponding to **PLANENUMBER**s that constitute the set:

VERTEXPLANESET(A,i) = [a,b,...d]

Define the set **TOTALPDVERTEXSET**(A,[a,b,...d]) as:

PDVERTEXSET(A,a) + **PDVERTEXSET**(A,b) + ... + **PDVERTEXSET**(A,d)

If i, i=1,2,...k, cannot be found such that the corresponding set [a,b,...d] satisfies:

TOTALPDVERTEXSET(A,[a,b,...d]) = **POLYVERTEXSET**(B) (3)

then polyhedrons A and B intersect by Theorem 2.1 and therefore, the relative positioning of polyhedrons A and B is unsafe.

If an i can be found such that the corresponding set [a,b,...d] satisfies (3), then the relative positioning of polyhedrons A and B is indeterminant at this point.

2.3.3.1 Theorem 2.1

If:

TOTALPDVERTEXSET(A,[a,b,...d]) <> **POLYVERTEXSET**(B) (4)

where [a,b,...d] = **VERTEXPLANESET**(A,i), i=1,2,...k

then polyhedrons A and B intersect.

Proof by contradiction:

Assume that polyhedrons A and B do not intersect. Since polyhedrons A and B are convex, then an infinite plane, p, may be found to divide all space into two regions, say I and II, such that polyhedrons A and B each lie entirely in regions I and II, respectively. This is expressed pictorially in figure 2.1. The figure shows a slice of polyhedrons A and B, which is not necessarily perpendicular to every sliced plane face, perpendicular to the infinite plane p.

To avoid confusion only the edges formed by the slice are shown and not the remaining parts of the plane faces they belong to.

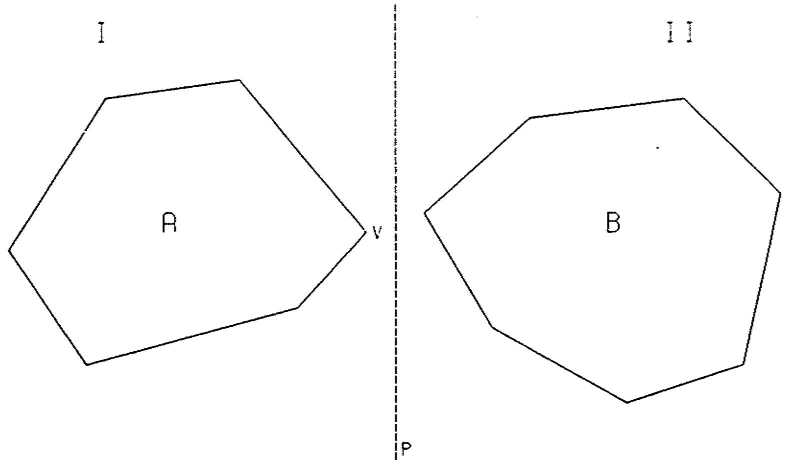


Figure 2.1 Slice of polyhedrons A and B

Notice that every point on the infinite plane, p, forms a positive displacement with at least one of the plane faces of polyhedron A, which share the common vertex of polyhedron A with **VERTEXNUMBER**=v. It necessarily follows that every vertex of polyhedron B forms a positive displacement with at least one of the plane faces of polyhedron A just mentioned. Thus, there exists at least one vertex of polyhedron A with **VERTEXNUMBER**=v such that (3) is satisfied. However, this contradicts (4). Therefore, the assumption that polyhedrons A and B do not intersect is incorrect. Thus, polyhedrons A and B intersect. This completes the proof.

2.3.4 Step 4

If the relative positioning of polyhedrons A and B is still indeterminant, then repeat step 3 but exchange the roles of polyhedrons A and B.

2.3.5 Step 5

If the relative positioning of polyhedrons A and B is still indeterminant, then for a plane face of polyhedron A with **PLANENUMBER**=j consider all the vertices corresponding to **VERTEXNUMBER**s that constitute the set:

PLANEVERTEXSET(A,j) = [a,b,...d]

Define the set **TOTALNDPLANESET**(A,[a,b,...d]) as:

NDPLANESET(A,a) + **NDPLANESET**(A,b) + ... + **NDPLANESET**(A,d)

If:

TOTALNDPLANESET(A,[a,b,...d]) <> **POLYPLANESET**(B) (5)

then no edge of the plane face of polyhedron A with **PLANENUMBER**=j can intersect polyhedron B by Theorem 2.2.

If no plane face of polyhedron A can be found to violate (5), then polyhedrons A and B do not intersect and therefore, the relative positioning of polyhedrons A and B is safe.

If one or more plane faces of polyhedron A can be found that violate (5), then the relative positioning of polyhedrons A and B is indeterminant at this point, however, the **PLANENUMBERs** of these plane faces form the following set:

PLANESET(A) = [e,f,...h]

The edges formed by adjacent plane faces with **PLANENUMBERs** belonging to the above set are the only edges of polyhedron A that can possibly intersect polyhedron B by Theorem 2.3. Retain this set for possible use in step 7.

2.3.5.1 Theorem 2.2

If:

TOTALNDPLANESET(A,[a,b,...d]) <> **POLYPLANESET**(B) (5)

where [a,b,...d] = **PLANEVERTEXSET**(A,j)

then no edge of the plane face of polyhedron A with **PLANENUMBER**=j can intersect polyhedron B.

Proof:

The inequality in (5) implies the existance of at least one plane face of polyhedron B with **PLANENUMBER**=q such that:

q **IN NDPLANESET**(A,a) = FALSE

q **IN NDPLANESET**(A,b) = FALSE

"

"

"

q **IN NDPLANESET**(A,d) = FALSE

that is, vertices of polyhedron A with **VERTEXNUMBERs** a,b,...d all have positive displacements with the plane face of polyhedron B with **PLANENUMBER**=q. Therefore, all points along the edges formed by these vertices have positive displacements with the plane face of polyhedron B with **PLANENUMBER**=q. Thus, no edge of the plane face of polyhedron A with **PLANENUMBER**=j can intersect polyhedron B. This completes the proof.

2.3.5.2 Theorem 2.3

Only edges of polyhedron A formed by adjacent plane faces with **PLANENUMBERs** belonging to the set **PLANESET**(A) = [e,f,...h] can possibly intersect polyhedron B.

Proof:

Define the set of all **PLANENUMBERs** corresponding to plane faces of polyhedron A which satisfy (5) as:

SAFEPLANESET(A) = [w,x,...z]

Since the set **PLANESET**(A) consists of all **PLANENUMBERs** corresponding to plane faces of polyhedron A which violate (5), then:

PLANESET(A) + **SAFEPLANESET**(A) = **POLYPLANESET**(A)

Therefore, all edges of polyhedron A are formed between adjacent plane faces with **PLANENUMBERs** either:

i) both belonging to **PLANESET**(A), or

ii) both belonging to **SAFEPLANESET**(A), or

iii) one belonging to **PLANESET**(A) and one to **SAFEPLANESET**(A)

Since all edges of all plane faces with **PLANENUMBERs** belonging to **SAFEPLANESET**(A) cannot intersect polyhedron B, then all edges corresponding to case **ii)** cannot intersect polyhedron B.

Any edge corresponding to case **iii)** belongs to a plane face with a **PLANENUMBER** belonging to **SAFEPLANESET**(A) and therefore, cannot intersect polyhedron B.

Therefore, the only edges of polyhedron A that can possibly intersect polyhedron B are those corresponding to case **i)**. Thus, only edges of polyhedron A formed by adjacent plane faces with **PLANENUMBERs** belonging to the set **PLANESET**(A) can possibly intersect polyhedron B. This completes the proof.

2.3.6 Step 6

If the relative positioning of polyhedrons A and B is still indeterminant, then repeat step 5 but exchange the roles of polyhedrons A and B.

2.3.7 Step 7

If the relative positioning of polyhedrons A and B is still indeterminant, then if an intersection does occur it will involve edges of polyhedrons A and B formed by adjacent plane faces with **PLANENUMBER**s belonging to the sets **PLANESET**(A) and **PLANESET**(B), respectively. These sets were generated in steps 5 and 6.

Among the edges formed by adjacent plane faces of polyhedron A and B with **PLANENUMBER**s belonging to the sets **PLANESET**(A) and **PLANESET**(B), respectively, there may still exist edges with endpoints both having positive displacements with the same plane face of the other polyhedron. Such an edge, o-p, of polyhedron A satisfies:

NDPLANESET(A,o) + **NDPLANESET**(A,p) <> **POLYPLANESET**(B)

and such an edge, o'-p', of polyhedron B satisfies:

NDPLANESET(B,o') + **NDPLANESET**(B,p') <> **POLYPLANESET**(A)

All points along such edges of a polyhedron have positive displacements with a plane face belonging to the other polyhedron and therefore, cannot possibly intersect the other polyhedron.

Any edge, u-v, of polyhedron A that can possibly intersect polyhedron B must satisfy:

PLANEVERTEXSET(A,s) * **PLANEVERTEXSET**(A,t) = [u,v]

&

NDPLANESET(A,u) + **NDPLANESET**(A,v) = **POLYPLANESET**(B) (6)

where s **IN PLANESET**(A) = TRUE & t **IN PLANESET**(A) = TRUE

Similarly any edge, u'-v', of polyhedron B that can possibly intersect polyhedron A must satisfy:

PLANEVERTEXSET(B,s') * **PLANEVERTEXSET**(B,t') = [u',v']

&

NDPLANESET(B,u') + **NDPLANESET**(B,v') = **POLYPLANESET**(A) (7)

where s' **IN PLANESET**(B) = TRUE & t' **IN PLANESET**(B) = TRUE

If an edge pair u-v of polyhedron A and u'-v' of polyhedron B satisfying (6) and (7), respectively, can be found such that the following conditions are met:

i) The edge u-v of polyhedron A straddles the plane faces of polyhedron B with **PLANENUMBER**s s' & t':

u **IN PDVERTEXSET**(B,s') = TRUE

u **IN PDVERTEXSET**(B,t') = FALSE

v **IN PDVERTEXSET**(B,s') = FALSE

v **IN PDVERTEXSET**(B,t') = TRUE

ii) The edge u'-v' of polyhedron B straddles the plane faces of polyhedron A with **PLANENUMBER**s s & t:

u' **IN PDVERTEXSET**(A,s) = TRUE

u' **IN PDVERTEXSET**(A,t) = FALSE

v' **IN PDVERTEXSET**(A,s) = FALSE

v' **IN PDVERTEXSET**(A,t) = TRUE

iii) The vertex of polyhedron A with **VERTEXNUMBER**=u has a positive displacement with the plane:

a) defined by the three vertices with **VERTEXNUMBERs** u' & v' of polyhedron B and v of polyhedron A.

b) and with the direction of its outward unit normal vector such that it has negative displacements with all vertices of polyhedron B with **VERTEXNUMBERs** not equal to u' or v'.

iv) The vertex of polyhedron B with **VERTEXNUMBER**=u' has a positive displacement with the plane:

a) defined by the three vertices with **VERTEXNUMBERs** u & v of polyhedron A and v' of polyhedron B.

b) and with the direction of its outward unit normal vector such that it has negative displacements with all vertices of polyhedron A with **VERTEXNUMBERs** not equal to u or v.

then polyhedrons A and B do not intersect by Theorem 2.4 . Therefore, the relative positioning of polyhedrons A and B is safe.

If no edge pair u-v of polyhedron A and u'-v' of polyhedron B satisfying (6) and (7), respectively, can be found to meet conditions **i)** through **iv)**, then the relative positioning of polyhedrons A and B is indeterminant, in which case it must be taken as unsafe.

2.3.7.1 Theorem 2.4

If there exists an edge pair u-v of polyhedron A and u'-v' of polyhedron B satisfying (6) and (7), respectively, that meets conditions **i)** through **iv)** on pages 20 and 21, then polyhedrons A and B do not intersect.

Proof:

Consider figure 2.2. This figure shows a slice of polyhedron B, similar to that of polyhedrons A and B in figure 2.1, which possesses the edge u-v of polyhedron A and is perpendicular to any plane containing the edge u'-v' of polyhedron B. The infinite planes containing the plane faces of polyhedron B with **PLANENUMBER**s s' & t' divide all space into the regions I through IV. Since condition **i)** is met the endpoints of u-v lie in regions II and IV. Since condition **iii)** is met u-v must lie entirely outside of region III.

Consider figure 2.3. This figure shows a slice of polyhedron A, similar to that of polyhedrons A and B in figure 2.1, which possesses the edge u'-v' of polyhedron B and is perpendicular to any plane containing the edge u-v of polyhedron A. The infinite planes containing the plane faces of polyhedron A with **PLANENUMBER**s s & t divide all space into the regions I' through IV'. Since condition **ii)** is met the endpoints of u'-v' lie in regions II' and IV'. Since condition **iv)** is met u'-v' must lie entirely outside of region III'.

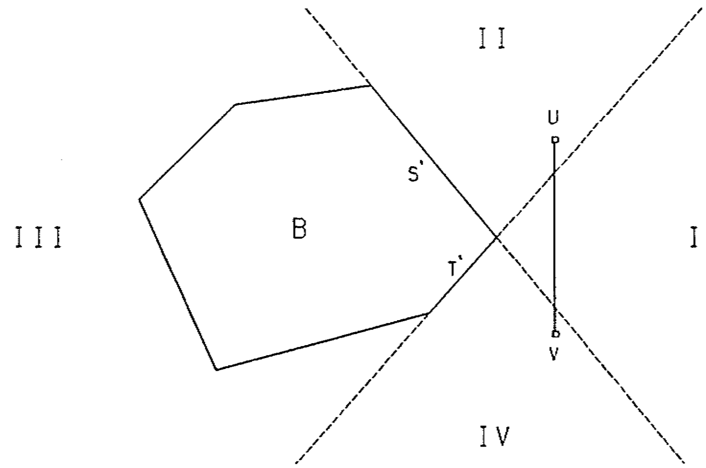


Figure 2.2 Slice of polyhedron B & edge u-v

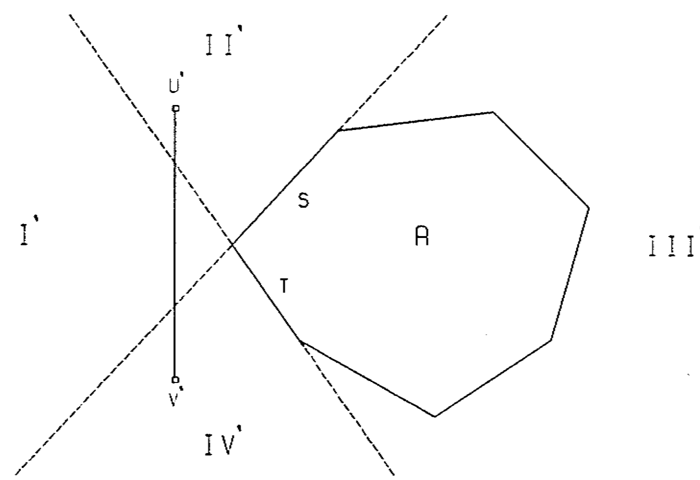


Figure 2.3 Slice of polyhedron A & edge u'-v'

Define the vectors **Va** and **Vb** as being parallel to the edges u-v and u'-v', respectively, and represent the cross-product between these two vectors as **Vc**. Define the infinite planes **PLANEA** and **PLANEB** as containing the edges u-v and u'-v', respectively, and such that the vector **Vc** is normal to both.

Consider figures 2.4 and 2.5. These figures are the same as figures 2.2 and 2.3 respectively except for the edges of the infinite planes **PLANEA** AND **PLANEB** which divide all space into the three regions I" through III". Since condition **i)** is met polyhedron B must lie entirely in region I" and since condition **ii)** is met polyhedron A must lie entirely in region III".

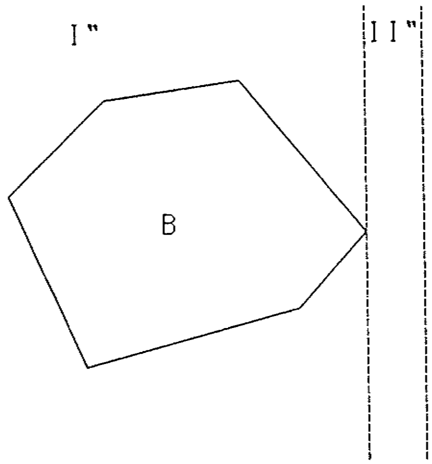


Figure 2.4 Slice of polyhedron B in region I"

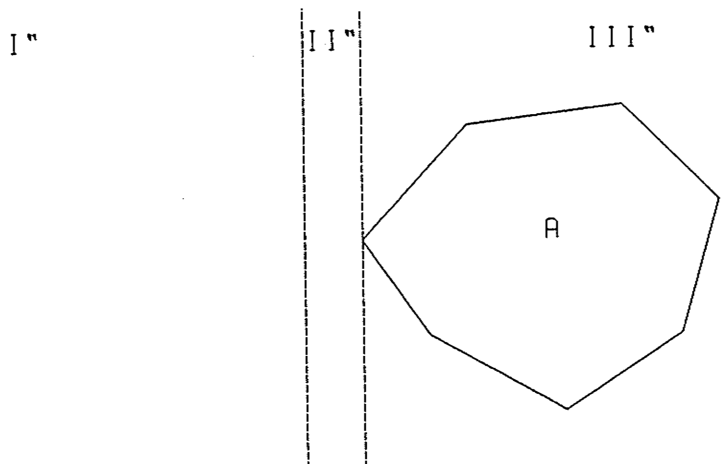


Figure 2.5 Slice of polyhedron A in region III"

Since polyhedrons A and B lie entirely in regions I" and III", respectively, and region II" separates regions I" and

III", then polyhedrons A and B do not intersect. This completes the proof.

2.4 Summary

The seven steps and the four theorems they employ constitute the intersection detection algorithm. This algorithm will be used in the iterative procedure of Chapter 4 to determine whether or not the sample configurations of a path are safe.

CHAPTER 3

Representation of Workspace and Manipulator using Linked Lists

This chapter presents the data structures used to represent the workspace, that is all obstacles and workpieces, and the manipulator in the Pascal program FINDPATH in Appendix A.

3.1 Records

A record is a data structure provided by the Pascal language to group together related data of different types. For example the following is a record used in the program FINDPATH to describe a plane face of a polyhedron:

```
PLANE=RECORD
 PLANENUMBER:NUMBER;
 OUTWARDNORMAL:VECTOR;
 NUMOFVERTICES:0..N;
 PLANEVERTEXSET,PDVERTEXSET:NUMBERSET;
 PLANEVERTEX:POINTPLANEVERTEX;
 NEXTPLANE:POINTPLANE
 END;
```

The identifiers PLANENUMBER, OUTWARDNORMAL, ... which are of the data types NUMBER, VECTOR, ..., respectively, specify the fields of the record.

3.2 Linked Lists

A linked list consists of a sequence of records where the first record is referenced by a Top pointer and each subsequent record is referenced by a pointer occupying a field of the previous record. This is illustrated in figure 3.1.

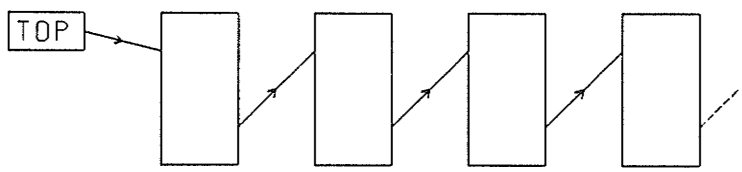


Figure 3.1 A linked list

Linked lists are used throughout the program FINDPATH instead of arrays due to the following three advantages of the former over the latter:

1) The size, or number of records, of a linked list is determined dynamically. Thus, during execution of a program a linked list may be created, have its size increased or decreased, and even be deleted entirely. Therefore a linked list uses only as much memory as required.

2) Since a record can contain more than one data type a single linked list can contain more than one data type.

3) Records can be inserted or deleted any where in a linked list. The insertion of a record R is shown by

steps 1 to 3 in figure 3.2. To delete the record R from the linked list of figure 3.2 simply follow steps 1 to 3 in reverse order.

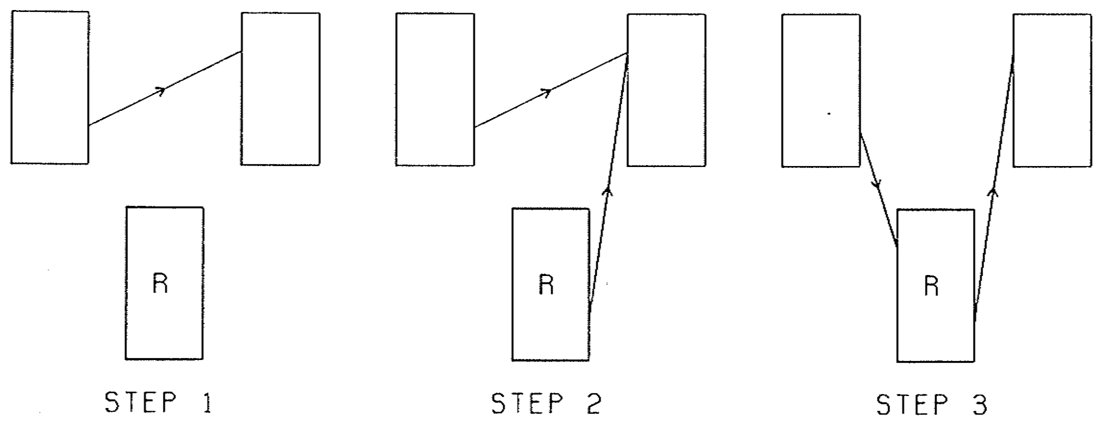


Figure 3.2 Insertion of a record into a linked list

The only disadvantage of linked lists in comparison to arrays is that in order to access a record all previous records must first be traversed.

3.3 Polyhedron

A polyhedron is represented by a record which contains among its fields pointers to a linked list of vertex records and to a linked list of plane records.

Each record in the linked list of vertex records corresponds to a vertex of the polyhedron.

Each record in the linked list of plane records corresponds to a plane face of the polyhedron and contains a pointer to a linked list of plane vertex records.

Each record in the linked list of plane vertex records corresponds to a vertex belonging to the plane face. The ordering of the plane vertex records within the linked list corresponds to either a clockwise or counter clockwise traversal of the edges belonging to the plane face.

The data structure for a polyhedron, described verbally in the preceding paragraphs, is shown in figure 3.3. In this figure the record types are identified by P, V, PL and PLV which stand for polyhedron, vertex, plane and plane vertex respectively.

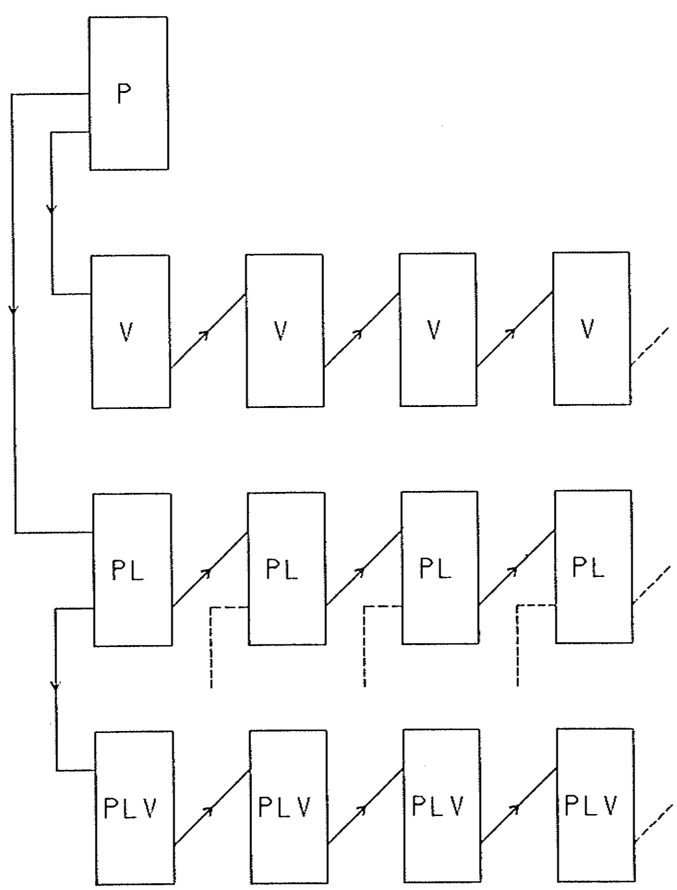


Figure 3.3 Data structure for a polyhedron

The algorithm presented in Chapter 2 required data defining the structure, location and orientation of each polyhedron and generated data defining the relative positioning of any two convex polyhedrons. The fields of each record within the polyhedron's data structure relevant to the data required and generated by the algorithm follow, listed under the records they belong to:

i) Polyhedron record

a) **POLYVERTEXSET**

b) **POLYPLANESET**

ii) Vertex record

a) **VERTEXNUMBER**

b) VERTEX

c) **VERTEXPLANESET**

d) **NDPLANESET**

iii) Plane record

a) **PLANENUMBER**

b) OUTWARDNORMAL

c) **PLANEVERTEXSET**

d) **PDVERTEXSET**

iv) Plane vertex record

a) **VERTEXNUMBER**

The above fields in bold type were all defined in Sections 2.1 and 2.3.1 of Chapter 2 using the same identifiers. The fields VERTEX and OUTWARDNORMAL are described by **VII** and **VIII**, respectively, of Section 2.1 of Chapter 2.

3.4 Obstacles

All obstacles in the workspace are organized as a linked list of polyhedron records.

Since the vertices and plane faces for each obstacle are read in by the program FINDPATH tests are required to ensure that all vertices belonging to a plane face are in fact coplanar and to ensure that the obstacle is in fact convex.

The coplanar test used by the program FINDPATH considers all vertices belonging to a plane face as coplanar if the inner product of a unit normal vector to the plane, defined by three vertices belonging to the plane face, and a vector formed from one of these vertices and any other vertex belonging to the plane face, is approximately zero.

The convexity test used by the program FINDPATH considers an obstacle as convex if for any plane face all vertices not belonging to the plane face have negative displacements with it, where displacement is the same as that defined in Section 2.1 of Chapter 2.

3.5 Workpieces

Each workpiece is simply a box that may assume any position and orientation on any plane parallel to the floor plane.

The dimensions, position and orientation of a workpiece occupy fields of a record referred to as DEFINEBOX. One of DEFINEBOX's fields contains a pointer to a polyhedron record, which defines all the vertices and plane faces of the workpiece for its particular position and orientation.

All workpieces in the workspace are organized as a linked list of DEFINEBOX records.

3.6 Manipulator

The manipulator is represented by a record, which contains among its fields pointers to records which define the polyhedrons that constitute the manipulator.

The pointers BASE, LINK3, WRIST, FINGER1, FINGER2 and, if a workpiece is being held, WORKPIECE all point to DEFINEBOX records while the pointers LINK2 and LINK3 point to DEFINELINK records. A DEFINELINK record represents a tapered box in any position and orientation on any plane perpendicular to the floor plane.

The positions and orientations of the polyhedrons referenced from the DEFINEBOX and DEFINELINK records are determined by fields of the manipulator record which specify the values of the manipulator's degrees of freedom. Thus one manipulator record specifies one configuration of the manipulator and, as will be seen in Chapter 4, a linked list of manipulator records can be used to specify a path.

CHAPTER 4

An Iterative Solution to the Find Path Problem

This chapter presents a solution to the find-path problem consisting of an iterative procedure, which employs the algorithm presented in Chapter 2, applied to an initial linear path.

4.1 Prerequisites and Assumptions

Any task given to a manipulator involves movement of a workpiece from one location to another. Thus, manipulator configurations may be placed into two categories. The first category contains all the configurations involved in grasping or releasing a workpiece and the second contains all other configurations. Refer to the former category as fixed configurations and the latter as variable configurations.

The path or paths required to execute a task will each consist of fixed and variable configurations. The initial linear path referred to in this chapter is actually the subpath consisting of only variable configurations and is bound by fixed configurations, referred to as start and goal configurations.

The manipulator simulated by the program FINDPATH has five degrees of freedom, namely:

1) Degree of Freedom Angle 1 - The angle formed between the base and a reference axis in the floor plane. This degree of freedom allows for rotation of the manipulator through 360 degrees with the exception of a 10 degree deadband.

2) Degree of Freedom Angle 2 - The angle formed between the first link and a vertical axis through the center of the base. This degree of freedom may assume positive and negative values.

3) Degree of Freedom Angle 3 - The angle formed between the second link and an axis through the center of the first link. This degree of freedom may assume positive values only.

4) Orientation - The angle formed between the projection of the wrist on the floor plane and the same axis mentioned in 1). This degree of freedom allows for rotation of the wrist.

5) Gap - The distance between the two fingers. This degree of freedom allows for grasping and releasing workpieces.

Reduction of the absolute value of the second degree of freedom raises all polyhedrons of the manipulator except the base. Reducing the third degree of freedom raises all polyhedrons except the base and the first link.

The procedure assumes that the manipulator's base is always safe and requires that the start and goal configurations have different first degree of freedom values. Assume all obstacles and workpieces rest on the floor plane since for such a workspace any unsafe configuration may be made safe by elevating polyhedrons of the manipulator.

4.2 Representation of the Initial Linear Path

The initial linear path is defined as the sequence of configurations between the start and goal configurations such that all degrees of freedom vary linearly. The number of configurations in this sequence is obviously infinite and therefore, the path must be represented by judiciously chosen sample configurations.

For convenience refer to the range of first degree of freedom values between those of the start and goal configurations as the path range. For any value in the path range let K, where $0 < K < 1$, be proportionate to the position of this value in the path range. For example if the values of degree of freedom angle 1 for the start and goal configurations are 10 and 50 degrees respectively, a sample configuration with a degree of freedom angle 1 value of 20 degrees would have a K value of 0.25.

One approach for choosing sample configurations might be to select those configurations with first degree of freedom values uniformly spaced in the path range. Such a selection of sample configurations would result in a sequence of K values that increase uniformly. This K value sequence is illustrated in figure 4.1.

Figure 4.1 K values for uniformly selected samples

This approach has two disadvantages. The first is that an obstacle or workpiece may lie entirely between two successive sample configurations. In this case all the sample configurations may be safe while the continuous path they represent is not. The second is that some of the sample configurations may never be able to intersect any obstacle or workpiece. In these cases all degrees of freedom other than the first may assume any value and therefore, consideration of such configurations is unnecessary.

An alternative approach which does not suffer from the above disadvantages is one where the workspace determines the selection of sample configurations. This approach may be described as follows:

1) Construct a linked list of the two manipulator records corresponding to the start and goal configurations.

2) For each obstacle there are two ranges of degree of freedom angle 1 that the obstacle lies in. These two ranges correspond to positive and negative values of degree of freedom angle 2. Determine these ranges and refer to them as the obstacle ranges.

3) For each workpiece determine the two ranges of degree of freedom angle 1 it lies in. Refer to these ranges as the workpiece ranges.

4) For each obstacle determine whether or not either of the obstacle ranges and the path range overlap. If so, select a sample configuration with a first degree of freedom value in the middle of the overlap range. Determine the value of K for this sample configuration and use it to insert the sample configuration's manipulator record between the appropriate record pair of the linked list mentioned in 1). If the overlap range is large enough, then select additional sample configurations at regular intervals throughout the overlap range but only as required since sample configurations may already exist within this range due to other obstacles and workpieces. Insert the manipulator records corresponding to these additional sample con-

figurations into the linked list using the previously mentioned procedure.

5) Repeat 4) but for workpieces not involved in the pick and place operations.

The sample configurations selected by this alternative approach would have a sequence of K values like that illustrated in figure 4.2.

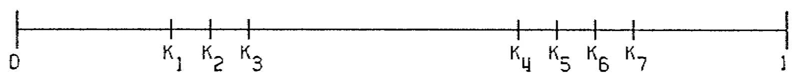


Figure 4.2 K values for workspace determined samples

The linked list, which contains the manipulator records corresponding to the sample configurations selected, represents all portions of the initial linear path in danger of collisions.

4.3 Iterative procedure

For the initial linear path determine which sample configurations, if any, are unsafe.

Whether or not a sample configuration is unsafe is determined by applying the algorithm of Chapter 2 to each polyhedron of the manipulator, except the base, and all polyhedrons of the workspace with ranges containing the value of the sample configuration's first degree of freedom. If the relative positioning of any polyhedron pair is unsafe, then the sample configuration is unsafe.

For each unsafe sample configuration reduce the absolute value of degree of freedom angle 2 and if necessary the value of degree of freedom angle 3, each by incremental amounts.

For the new path determine whether or not any unsafe sample configurations exist. If so, repeat the process described in the above two paragraphs, otherwise the current path is the safe path sought.

To prevent drastic changes in values of the second and third degrees of freedom from one sample configuration to the next these values must be adjusted during each iteration. The adjustment process will only be described for the second degree of freedom since it is similar for the third.

For a sample configuration compare the value of the second degree of freedom with the value generated linearly from the second degree of freedom values of the previous and subsequent sample configurations. If the absolute value of the generated value is less than the absolute value of the current value, then replace the current value with the generated value. This process is illustrated in steps 1 to 3 of figure 4.3 where the codes P, C, S and G represent the previous, current, subsequent and generated values, respectively.

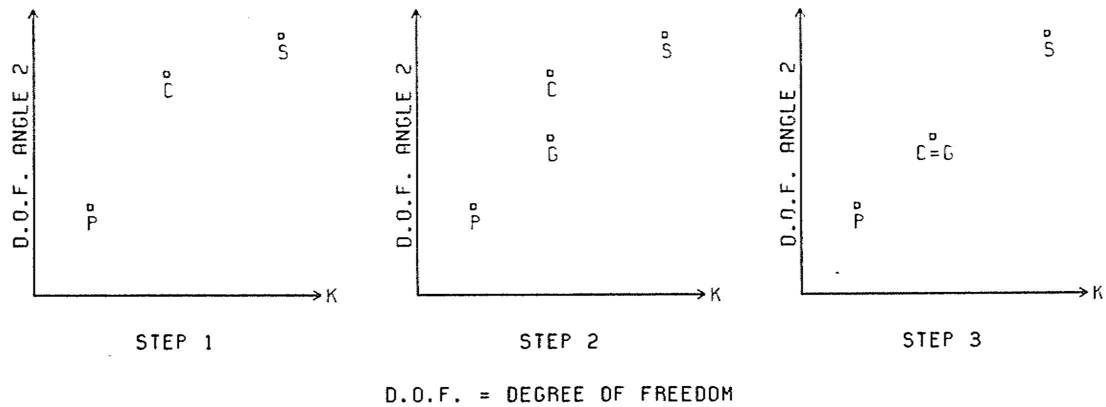


Figure 4.3 Reduction of differences between samples

4.4 Results

The output of program FINDPATH in Appendix B shows a typical task being performed by the manipulator. In order to execute this task safely three find-path problems had to be solved requiring under three minutes of execution time.

For each path, a pair of graphs is generated for each iteration required to produce the safe subpath. These graphs, consisting of plots of degree of freedom angles 2 and 3 both against K for variable configurations only, show the progression of the subpath from indeterminant and linear to safe and nonlinear.

All the fixed and variable configurations which constitute the path, as well as the workspace, are illustrated in the graphics following the set of graphs.

The program FINDPATH may accomplish any other task by altering the dataset TASK and in any other workspace by altering either or both of the datasets OBSTS and WKPCS.

CHAPTER 5

Summary and Conclusions

In this thesis an algorithm for detection of geometrical intersections in three dimensions was conceived, formulated and successfully implemented.

A problem oriented approach for selecting sample configurations of a continuous path was devised. This approach allows the workspace to dictate the choice of samples and results in a representation of all parts of the path in danger of collision and of only those parts.

An iterative procedure, incorporating the intersection detection algorithm and sample selection technique mentioned above, proved to be successful in solving the find-path problem for a hypothetical manipulator.

References

[1] Felix M. Arscott: Engineering Analysis I (Functional Analysis) Lecture Notes, University of Manitoba, 1984.

[2] J.A. Bate, M.S. Dole and H.J. Ferch: Computer Programming and Data Structures using Pascal, The Charles Babbage Research Centre, St. Pierre, MB., 1984.

[3] Rodney A. Brooks: Solving the Find-Path Problem by Good Representation of Free Space, "IEEE Transactions on Systems, Man, and Cybernetics", pp. 190-197, Vol. SMC-13, No. 3, March/April 1983.

[4] Steven D. Eppinger and Warren P. Seering: Introduction to Dynamic Models for Robot Force Control, "IEEE Control Systems Magazine", pp. 48-52, April 1987.

[5] L. Gouzenes: Strategies for Solving Collison-free Trajectories Problems for Mobile and Manipulator Robots, "The International Journal of Robotics Research", pp. 51-65, Vol. 3, No. 4, Winter 1984.

[6] Thomas Lozano-Perez: Spatial Planning: A Configuration Space Approach, "IEEE Transactions on Computers", pp. 108-120, Vol. C-32, No. 2, February 1983.

[7] Thomas Lozano-Perez: Automatic Planning of Manipulator Transfer Movements, "IEEE Transactions on Systems, Man, and Cybernetics", pp. 681-698, Vol. SMC-11, No. 10, October 1981.

[8] E. Mendelson: Introduction to Mathematical Logic, Van Nostrand Reinhold Co., New York, 1964.

[9] N. Nilsson: Problem-Solving Methods in Artificial Intelligence, McGraw-Hill Inc., New York, 1971.

[10] G.D. Sawatzky: An Efficient Collison Detector for Automatic Path Planning for Manipulators, M.Sc. Thesis, University of Manitoba, Winnipeg M.B., 1986.

[11] M. Vidyasagar: System Theory and Robotics, "IEEE Control Systems Magazine", pp. 16-17, April 1987.

APPENDIX A

Listing of Program FINDPATH

```
PROGRAM FINDPATH(INPUT,OUTPUT,OBSTS,BOX,WKPCS,TASK,ERRORS);

   ***********************************************************
   * WRITTEN BY  M.J. PROCCA - COPYRIGHT 1988  M.J. PROCCA *
   ***********************************************************

LABEL 1,2;

CONST
 N=20;
 MAX=100;
 PI=3.14159;
 THICKNESS=7.5;
 LBASE=20;   HBASE=15;
 LLINK1=55;  H1LINK1=12.5;  H2LINK1=10;
 LLINK2=50;  H1LINK2=10;  H2LINK2=7.5;
 LLINK3=7.5; HLINK3=7.5;
 LWRIST=5;   WWRIST=5;   HWRIST=1;
 LFINGER=5;  WFINGER=0.5; HFINGER=5;
 CLEARANCE=0.001;
 SAFETYFACTOR=2;
 LIFTHEIGHT=10;
 STARTPAGENUMBER=139;

TYPE

 POINT=ARRAY[1..3] OF REAL;
 PLANEPOINT=ARRAY[1..2] OF REAL;
 VECTOR=ARRAY[1..3] OF REAL;
 PLANEVECTOR=ARRAY[1..2] OF REAL;
 NUMBER=1..N;
 NUMBERSET=SET OF NUMBER;
 POLYHEDRANAMES=(OBSTACLE,FREEWORKPIECE,BASENAME,LINK1NAME,
                 LINK2NAME,LINK3NAME,WRISTNAME,FINGER1NAME,
                 FINGER2NAME,HELDWORKPIECE);
 POINTTYPES=(REGULAR,ADDED);
 EDGESTATUSNAMES=(VISIBLE,PARTVISIBLE,INVISIBLE);
 PENSTATUSNAMES=(RAISEPEN,LOWERPEN,HOLDPEN);
 ANGLES=ARRAY[1..3] OF REAL;
 HOLDSTATUSNAMES=(HOLDNEWWORKPIECE,HOLDOLDWORKPIECE,NOWORKPIECE);
 CHARS=PACKED ARRAY[1..40] OF CHAR;
 MANIPULATORTYPES=(FIXED,VARIABLE);

 POINTPLANEVERTEX=@PLANEVERTEX;
 PLANEVERTEX=RECORD
  GRAPHICSVERTEX:PLANEPOINT;
  PENDOWN:BOOLEAN;
  ADDPLANEVERTEX,NEXTPLANEVERTEX:POINTPLANEVERTEX;
  CASE POINTTYPE:POINTTYPES OF
  REGULAR:(VERTEXNUMBER:NUMBER;VERTEX:POINT;
  EDGESTATUS:EDGESTATUSNAMES;PENSTATUS:PENSTATUSNAMES);
  ADDED:(DISTANCE:REAL)
  END;
```

```
POINTPLANE=@PLANE;
PLANE=RECORD
 PLANENUMBER:NUMBER;
 OUTWARDNORMAL:VECTOR;
 NUMOFVERTICES:0..N;
 PLANEVERTEXSET,PDVERTEXSET:NUMBERSET;
 PLANEVERTEX:POINTPLANEVERTEX;
 NEXTPLANE:POINTPLANE
 END;

POINTVERTEX=@VERTEX;
VERTEX=RECORD
 VERTEXNUMBER:NUMBER;
 VERTEX:POINT;
 GRAPHICSVERTEX:PLANEPOINT;
 VERTEXPLANESET,NDPLANESET:NUMBERSET;
 NEXTVERTEX:POINTVERTEX
 END;

POINTPOLYHEDRON=@POLYHEDRON;
POLYHEDRON=RECORD
 POLYHEDRONTYPE:POLYHEDRANAMES;
 POLYHEDRONNUMBER:NUMBER;
 LOWERBOUND1,UPPERBOUND1,LOWERBOUND2,UPPERBOUND2:REAL;
 NORMALBOUNDS1,NORMALBOUNDS2:BOOLEAN;
 POLYVERTEXSET,POLYPLANESET,
 VISIBLEPLANESET:NUMBERSET;
 POLYHEDRONVERTEX:POINTVERTEX;
 NUMOFPLANES:0..N;
 FIRSTPLANE:POINTPLANE;
 NEXTPOLYHEDRON:POINTPOLYHEDRON
 END;

POINTDEFINEBOX=@DEFINEBOX;
DEFINEBOX=RECORD
 LENGTH,WIDTH,HEIGHT,
 HYPOTENUSE,BETA:REAL;
 REFERENCE:POINT;
 THETA:REAL;
 BOXPOLYHEDRON:POINTPOLYHEDRON;
 NEXTDEFINEBOX:POINTDEFINEBOX
 END;

POINTDEFINELINK=@DEFINELINK;
DEFINELINK=RECORD
 LENGTH,WIDTH,HEIGHT1,HEIGHT2,
 HYPOTENUSE1,HYPOTENUSE2,BETA1,BETA2:REAL;
 REFERENCE:POINT;
 THETA1,THETA2:REAL;
 BOXPOLYHEDRON:POINTPOLYHEDRON
 END;

POINTMANIPULATOR=@MANIPULATOR;
MANIPULATOR=RECORD
```

```
  MANIPULATORTYPE:MANIPULATORTYPES;
  K:REAL;
  OBSTACLESET,WORKPIECESET:NUMBERSET;
  SAFEMANIPULATOR:BOOLEAN;
  DOFANGLE:ANGLES;
  ORIENTATION,GAP:REAL;
  BASE,LINK3,WRIST,
  FINGER1,FINGER2,WORKPIECE:POINTDEFINEBOX;
  LINK1,LINK2:POINTDEFINELINK;
  NEXTMANIPULATOR:POINTMANIPULATOR
  END;

VAR

 I:1..3;
 NUMOFOBSTACLES,WKPCNUMBER,PREVWKPCNUMBER:0..N;
 PATHNUMBER,PAGENUMBER:INTEGER;
 ANGLE,DISTANCE:REAL;
 LOCATION:POINT;
 ALPHA:ANGLES;
 GRAPHICSBASIS:ARRAY[1..3] OF VECTOR;
 CODE:CHAR;
 MESSAGE:CHARS;
 WORKSPACEERROR,WORKPIECEERROR,MANIPULATORERROR,
 INITIALPLANE,FIRSTPATH,LASTPATH,SAFEPATH:BOOLEAN;
 CURRPOLYVERTEX,PREVPOLYVERTEX:POINTVERTEX;
 CURRPLANEVERTEX,PREVPLANEVERTEX:POINTPLANEVERTEX;
 CURRENTPLANE,PREVIOUSPLANE:POINTPLANE;
 FIRSTOBSTACLE,CURRPOLYHEDRON,PREVPOLYHEDRON:POINTPOLYHEDRON;
 FIRSTWORKPIECE,CURRWORKPIECE,PREVWORKPIECE,
 WKPCPOINTER,PREVWKPCPOINTER:POINTDEFINEBOX;
 CURRENTPATHSTEP,PREVIOUSPATHSTEP,
 PATHSTART,POSTPATHSTART,PATHPOINTER1,
 PATHPOINTER2,PREPATHGOAL,PATHGOAL:POINTMANIPULATOR;
 OBSTS,BOX,WKPCS,TASK,ERRORS:TEXT;

              ************************************
              * CALCOMP SUBROUTINE DECLARATIONS *
              ************************************

PROCEDURE AREA(CONST X1,X2:SHORTREAL);FORTRAN;
PROCEDURE PLOTS(CONST IBUF,NLOC,LDEV:INTEGER);FORTRAN;
PROCEDURE SYMBOL(CONST X1,X2,X3:SHORTREAL;VAR S1:CHARS;
                 CONST X4:SHORTREAL;CONST X5:INTEGER);FORTRAN;
PROCEDURE PLOT(CONST XPAGE,YPAGE:SHORTREAL;CONST PENUP:INTEGER);
               FORTRAN;

 ***************************************************************
 * THE ABOVE SUBROUTINES WERE PROVIDED BY COMPUTER SERVICES AT *
 * THE UNIVERSITY OF MANITOBA  AND CONSTITUTE THE ONLY PART OF *
 * THE PROGRAM NOT WRITTEN BY  M.J. PROCCA.                    *
 ***************************************************************

PROCEDURE PLANERECORDS(POINTER:POINTPOLYHEDRON;VAR INFILE:TEXT;
```

```
                        INITIAL:BOOLEAN);

VAR
 I:1..3;
 J:NUMBER;
 MAGNITUDE:REAL;
 VERTICES:ARRAY[1..3] OF POINT;
 VEC1,VEC2,NORMAL:VECTOR;

BEGIN
CODE:=' ';
WITH POINTER@ DO
 WHILE NOT EOF(INFILE) AND (CODE<>'*') DO
  IF NOT EOLN(INFILE) THEN

   BEGIN
   READ(INFILE,CODE);

   CASE CODE OF

   ' ':;

   'P':

    BEGIN
    IF INITIAL THEN

     BEGIN
     NUMOFPLANES:=NUMOFPLANES+1;
     NEW(CURRENTPLANE);

     WITH CURRENTPLANE@ DO

      BEGIN
      PLANENUMBER:=NUMOFPLANES;
      POLYPLANESET:=POLYPLANESET+[PLANENUMBER];
      NUMOFVERTICES:=0;
      PLANEVERTEXSET:=[];
      PLANEVERTEX:=NIL;
      PREVPLANEVERTEX:=NIL;
      NEXTPLANE:=NIL
      END;

     IF FIRSTPLANE=NIL THEN
      FIRSTPLANE:=CURRENTPLANE;
     IF PREVIOUSPLANE<>NIL THEN
      PREVIOUSPLANE@.NEXTPLANE:=CURRENTPLANE;
     PREVIOUSPLANE:=CURRENTPLANE
     END

    ELSE

     BEGIN
     IF INITIALPLANE THEN
```

```
   INITIALPLANE:=FALSE
  ELSE
   CURRENTPLANE:=CURRENTPLANE@.NEXTPLANE;
  CURRPLANEVERTEX:=CURRENTPLANE@.PLANEVERTEX
  END;

 J:=1
 END;

'V':
 WITH CURRENTPLANE@ DO

  BEGIN
  IF INITIAL THEN

   BEGIN
   NUMOFVERTICES:=NUMOFVERTICES+1;
   NEW(CURRPLANEVERTEX)
   END;

  WITH CURRPLANEVERTEX@ DO

   BEGIN
   IF EOLN(INFILE) THEN
    READLN(INFILE);
   READ(INFILE,VERTEXNUMBER);

   IF INITIAL THEN

    BEGIN
    PLANEVERTEXSET:=PLANEVERTEXSET+[VERTEXNUMBER];
    EDGESTATUS:=VISIBLE;
    PENSTATUS:=HOLDPEN;
    PENDOWN:=TRUE;
    ADDPLANEVERTEX:=NIL;
    NEXTPLANEVERTEX:=NIL
    END;

   CURRPOLYVERTEX:=POLYHEDRONVERTEX;

   WHILE CURRPOLYVERTEX<>NIL DO
    IF VERTEXNUMBER=CURRPOLYVERTEX@.VERTEXNUMBER THEN

     BEGIN
     VERTEX:=CURRPOLYVERTEX@.VERTEX;
     GRAPHICSVERTEX:=CURRPOLYVERTEX@.GRAPHICSVERTEX;
     CURRPOLYVERTEX@.VERTEXPLANESET:=CURRPOLYVERTEX@.
     VERTEXPLANESET+[PLANENUMBER];
     CURRPOLYVERTEX:=NIL
     END

    ELSE
     CURRPOLYVERTEX:=CURRPOLYVERTEX@.NEXTVERTEX;
```

```
    IF J<=3 THEN

     BEGIN
     VERTICES[J]:=VERTEX;
     IF J=3 THEN

      BEGIN
      FOR I:=1 TO 3 DO

       BEGIN
       VEC1[I]:=VERTICES[2,I]-VERTICES[1,I];
       VEC2[I]:=VERTICES[3,I]-VERTICES[1,I]
       END;

      NORMAL[1]:=VEC1[2]*VEC2[3]-VEC2[2]*VEC1[3];
      NORMAL[2]:=VEC2[1]*VEC1[3]-VEC1[1]*VEC2[3];
      NORMAL[3]:=VEC1[1]*VEC2[2]-VEC2[1]*VEC1[2];

      MAGNITUDE:=SQRT(SQR(NORMAL[1])+
      SQR(NORMAL[2])+SQR(NORMAL[3]));

      FOR I:=1 TO 3 DO
      OUTWARDNORMAL[I]:=NORMAL[I]/MAGNITUDE
      END

     END

    END;

   IF INITIAL THEN

    BEGIN
    IF PLANEVERTEX=NIL THEN
     PLANEVERTEX:=CURRPLANEVERTEX;
    IF PREVPLANEVERTEX<>NIL THEN
     PREVPLANEVERTEX@.NEXTPLANEVERTEX:=CURRPLANEVERTEX;
    PREVPLANEVERTEX:=CURRPLANEVERTEX
    END

   ELSE
    CURRPLANEVERTEX:=CURRPLANEVERTEX@.NEXTPLANEVERTEX;

   J:=J+1
   END;

  '*':

  END

 END

ELSE
 READLN(INFILE)
```

```
END;

FUNCTION INVTAN(X,Y:REAL):REAL;

CONST
 DELTA=0.0001;

BEGIN
IF ABS(X)<=DELTA THEN

 BEGIN
 IF Y>0 THEN
  INVTAN:=PI/2
 ELSE
  INVTAN:=-PI/2
 END

ELSE
 IF X>0 THEN

  BEGIN
  IF Y>0 THEN
   INVTAN:=ARCTAN(Y/X)
  ELSE
   INVTAN:=-ARCTAN(-Y/X)
  END

 ELSE
  IF Y>0 THEN
   INVTAN:=PI-ARCTAN(-Y/X)
  ELSE
   INVTAN:=ARCTAN(Y/X)-PI
END;

FUNCTION ADJUSTANGLE(THETA:REAL):REAL;

BEGIN
IF THETA>PI THEN
 ADJUSTANGLE:=THETA-2*PI
ELSE
 IF THETA<-PI THEN
  ADJUSTANGLE:=THETA+2*PI
 ELSE
  ADJUSTANGLE:=THETA
END;

PROCEDURE FINDBOUNDS(POINTER:POINTPOLYHEDRON;VAR FIRSTVERTEX:BOOLEAN);

VAR
 R1,R2,R3,GAMMA1,GAMMA2,GAMMA3,
 LOWERANGLE1,UPPERANGLE1,
 LOWERANGLE2,UPPERANGLE2,DELTALOWER,DELTAUPPER:REAL;

BEGIN
```

```
WITH POINTER@,CURRPOLYVERTEX@ DO

 BEGIN
 R1:=1.5*THICKNESS+CLEARANCE;
 R3:=SQRT(SQR(VERTEX[1])+SQR(VERTEX[2]));

 IF (R3<R1) OR (R3<SQRT(SQR(0.5*LBASE)+SQR(0.5*THICKNESS))) THEN

  BEGIN
  IF POLYHEDRONTYPE=OBSTACLE THEN
   WRITELN(ERRORS,'OBSTACLE NUMBER':15,POLYHEDRONNUMBER:4,
           ' IS TOO CLOSE TO BASE OF MANIPULATOR *':38)
  ELSE
   WRITELN(ERRORS,'WORKPIECE NUMBER':16,POLYHEDRONNUMBER:4,
           ' IS TOO CLOSE TO BASE OF MANIPULATOR *':38);
  GOTO 1
  END;

 R2:=SQRT(SQR(R3)-SQR(R1));
 GAMMA1:=ARCTAN(R1/R2);
 R1:=THICKNESS/2;
 R2:=SQRT(SQR(R3)-SQR(R1));
 GAMMA2:=ARCTAN(R1/R2);
 GAMMA3:=INVTAN(VERTEX[1],VERTEX[2]);
 LOWERANGLE1:=ADJUSTANGLE(GAMMA3-GAMMA1);
 UPPERANGLE1:=ADJUSTANGLE(GAMMA3+GAMMA2);
 LOWERANGLE2:=ADJUSTANGLE(GAMMA3-GAMMA2-PI);
 UPPERANGLE2:=ADJUSTANGLE(GAMMA3+GAMMA1-PI);

 IF FIRSTVERTEX THEN

  BEGIN
  FIRSTVERTEX:=FALSE;
  LOWERBOUND1:=LOWERANGLE1;
  UPPERBOUND1:=UPPERANGLE1;
  LOWERBOUND2:=LOWERANGLE2;
  UPPERBOUND2:=UPPERANGLE2
  END

 ELSE

  BEGIN
  DELTALOWER:=LOWERANGLE1-LOWERBOUND1;
  IF ((-PI<DELTALOWER) AND (DELTALOWER<0)) OR (PI<DELTALOWER) THEN
   LOWERBOUND1:=LOWERANGLE1;
  DELTAUPPER:=UPPERANGLE1-UPPERBOUND1;
  IF ((0<DELTAUPPER) AND (DELTAUPPER<PI)) OR (DELTAUPPER<-PI) THEN
   UPPERBOUND1:=UPPERANGLE1;
  DELTALOWER:=LOWERANGLE2-LOWERBOUND2;
  IF ((-PI<DELTALOWER) AND (DELTALOWER<0)) OR (PI<DELTALOWER) THEN
   LOWERBOUND2:=LOWERANGLE2;
  DELTAUPPER:=UPPERANGLE2-UPPERBOUND2;
  IF ((0<DELTAUPPER) AND (DELTAUPPER<PI)) OR (DELTAUPPER<-PI) THEN
   UPPERBOUND2:=UPPERANGLE2
```

```
  END

 END

END;

PROCEDURE CLASSIFYBOUNDS(POINTER:POINTPOLYHEDRON);

VAR
 SAVEBOUND:REAL;

BEGIN
WITH POINTER@ DO

 BEGIN
 IF LOWERBOUND1<UPPERBOUND1 THEN
  NORMALBOUNDS1:=TRUE
 ELSE

  BEGIN
  NORMALBOUNDS1:=FALSE;
  SAVEBOUND:=LOWERBOUND1;
  LOWERBOUND1:=UPPERBOUND1;
  UPPERBOUND1:=SAVEBOUND
  END;

 IF LOWERBOUND2<UPPERBOUND2 THEN
  NORMALBOUNDS2:=TRUE
 ELSE

  BEGIN
  NORMALBOUNDS2:=FALSE;
  SAVEBOUND:=LOWERBOUND2;
  LOWERBOUND2:=UPPERBOUND2;
  UPPERBOUND2:=SAVEBOUND
  END

 END

END;

PROCEDURE BOXRECORDS(POINTER:POINTDEFINEBOX;INITIAL:BOOLEAN);

VAR
 I:1..9;
 J:1..5;
 GAMMA:ARRAY[1..4] OF REAL;
 FIRSTVERTEX:BOOLEAN;

PROCEDURE POLYVERTEXRECORD(LAMBDA:REAL);

VAR
 K:1..3;
 VEC:VECTOR;
```

```
BEGIN
WITH POINTER@ DO

 BEGIN
 IF INITIAL THEN
  NEW(CURRPOLYVERTEX);

 WITH CURRPOLYVERTEX@ DO

  BEGIN
  VERTEX[1]:=REFERENCE[1]+HYPOTENUSE*COS(GAMMA[J]);
  VERTEX[2]:=REFERENCE[2]+HYPOTENUSE*SIN(GAMMA[J]);
  VERTEX[3]:=LAMBDA;
  FOR K:=1 TO 3 DO
   VEC[K]:=VERTEX[K];
  FOR K:=1 TO 2 DO
   GRAPHICSVERTEX[K]:=VEC[1]*GRAPHICSBASIS[K,1]+VEC[2]*
   GRAPHICSBASIS[K,2]+VEC[3]*GRAPHICSBASIS[K,3];
  IF BOXPOLYHEDRON@.POLYHEDRONTYPE=FREEWORKPIECE THEN
   FINDBOUNDS(BOXPOLYHEDRON,FIRSTVERTEX)
  END;

 IF INITIAL THEN
  WITH BOXPOLYHEDRON@ DO

   BEGIN
   WITH CURRPOLYVERTEX@ DO

    BEGIN
    VERTEXNUMBER:=I;
    POLYVERTEXSET:=POLYVERTEXSET+[VERTEXNUMBER];
    VERTEXPLANESET:=[];
    NEXTVERTEX:=NIL
    END;

   IF POLYHEDRONVERTEX=NIL THEN
    POLYHEDRONVERTEX:=CURRPOLYVERTEX;
   IF PREVPOLYVERTEX<>NIL THEN
    PREVPOLYVERTEX@.NEXTVERTEX:=CURRPOLYVERTEX;
   PREVPOLYVERTEX:=CURRPOLYVERTEX
   END

  ELSE
   CURRPOLYVERTEX:=CURRPOLYVERTEX@.NEXTVERTEX

 END

END;

BEGIN
WITH POINTER@ DO

 BEGIN
```

```
WITH BOXPOLYHEDRON@ DO
 IF INITIAL THEN

  BEGIN
  POLYVERTEXSET:=[];
  POLYHEDRONVERTEX:=NIL;
  PREVPOLYVERTEX:=NIL;
  POLYPLANESET:=[];
  NUMOFPLANES:=0;
  VISIBLEPLANESET:=[];
  FIRSTPLANE:=NIL;
  PREVIOUSPLANE:=NIL
  END

 ELSE

  BEGIN
  CURRPOLYVERTEX:=POLYHEDRONVERTEX;
  CURRENTPLANE:=FIRSTPLANE;
  INITIALPLANE:=TRUE
  END;

GAMMA[1]:=THETA+BETA;
GAMMA[2]:=THETA-BETA;
GAMMA[3]:=GAMMA[1]-PI;
GAMMA[4]:=GAMMA[2]-PI;
I:=1;
J:=1;
FIRSTVERTEX:=TRUE;

WHILE I<=4 DO

 BEGIN
 POLYVERTEXRECORD(REFERENCE[3]);
 I:=I+1;
 J:=J+1
 END;

J:=1;

WHILE I<=8 DO

 BEGIN
 POLYVERTEXRECORD(REFERENCE[3]+HEIGHT);
 I:=I+1;
 J:=J+1
 END;

IF BOXPOLYHEDRON@.POLYHEDRONTYPE=FREEWORKPIECE THEN
 CLASSIFYBOUNDS(BOXPOLYHEDRON);
RESET(BOX);
PLANERECORDS(BOXPOLYHEDRON,BOX,INITIAL)
END
```

```
END;

PROCEDURE LINKRECORDS(POINTER:POINTDEFINELINK;INITIAL:BOOLEAN);

VAR
 I:1..8;
 J:1..3;
 R1,R2,R3,GAMMA:REAL;
 VEC:VECTOR;

BEGIN
WITH POINTER@ DO

 BEGIN
 WITH BOXPOLYHEDRON@ DO
  IF INITIAL THEN

   BEGIN
   POLYVERTEXSET:=[];
   POLYHEDRONVERTEX:=NIL;
   PREVPOLYVERTEX:=NIL;
   POLYPLANESET:=[];
   NUMOFPLANES:=0;
   VISIBLEPLANESET:=[];
   FIRSTPLANE:=NIL;
   PREVIOUSPLANE:=NIL
   END

  ELSE

   BEGIN
   CURRPOLYVERTEX:=POLYHEDRONVERTEX;
   CURRENTPLANE:=FIRSTPLANE;
   INITIALPLANE:=TRUE
   END;

 R1:=WIDTH/2;

 FOR I:=1 TO 8 DO

  BEGIN
  IF INITIAL THEN
   NEW(CURRPOLYVERTEX);

  WITH CURRPOLYVERTEX@ DO

   BEGIN
   CASE I OF

   1,2:

    BEGIN
    IF I=1 THEN
```

```
   BEGIN
   R2:=HYPOTENUSE2*COS(THETA2-BETA2);
   R3:=SQRT(SQR(R1)+SQR(R2));
   IF R2>0 THEN
    GAMMA:=PI/2-ARCTAN(R2/R1)
   ELSE
    GAMMA:=PI/2+ARCTAN(-R2/R1)
   END

  ELSE
   GAMMA:=-GAMMA;

  VERTEX[1]:=REFERENCE[1]+R3*COS(THETA1+GAMMA);
  VERTEX[2]:=REFERENCE[2]+R3*SIN(THETA1+GAMMA);
  VERTEX[3]:=REFERENCE[3]+HYPOTENUSE2*
  SIN(THETA2-BETA2)
  END;

 3,4:

  BEGIN
  IF I=3 THEN

   BEGIN
   R2:=HYPOTENUSE1*COS(THETA2+BETA1-PI);
   R3:=SQRT(SQR(R1)+SQR(R2));
   IF R2>0 THEN
    GAMMA:=-PI/2+ARCTAN(R2/R1)
   ELSE
    GAMMA:=-PI/2-ARCTAN(-R2/R1)
   END

  ELSE
   GAMMA:=-GAMMA;

  VERTEX[1]:=REFERENCE[1]+R3*COS(THETA1+GAMMA);
  VERTEX[2]:=REFERENCE[2]+R3*SIN(THETA1+GAMMA);
  VERTEX[3]:=REFERENCE[3]+HYPOTENUSE1*
  SIN(THETA2+BETA1-PI)
  END;

 5,6:

  BEGIN
  IF I=5 THEN

   BEGIN
   R2:=HYPOTENUSE2*COS(THETA2+BETA2);
   R3:=SQRT(SQR(R1)+SQR(R2));
   IF R2>0 THEN
    GAMMA:=PI/2-ARCTAN(R2/R1)
   ELSE
    GAMMA:=PI/2+ARCTAN(-R2/R1)
   END
```

```
 ELSE
  GAMMA:=-GAMMA;

 VERTEX[1]:=REFERENCE[1]+R3*COS(THETA1+GAMMA);
 VERTEX[2]:=REFERENCE[2]+R3*SIN(THETA1+GAMMA);
 VERTEX[3]:=REFERENCE[3]+HYPOTENUSE2*
 SIN(THETA2+BETA2)
 END;

7,8:

 BEGIN
 IF I=7 THEN

  BEGIN
  R2:=HYPOTENUSE1*COS(THETA2-BETA1+PI);
  R3:=SQRT(SQR(R1)+SQR(R2));
  IF R2>0 THEN
   GAMMA:=-PI/2+ARCTAN(R2/R1)
  ELSE
   GAMMA:=-PI/2-ARCTAN(-R2/R1)
  END

 ELSE
  GAMMA:=-GAMMA;

 VERTEX[1]:=REFERENCE[1]+R3*COS(THETA1+GAMMA);
 VERTEX[2]:=REFERENCE[2]+R3*SIN(THETA1+GAMMA);
 VERTEX[3]:=REFERENCE[3]+HYPOTENUSE1*
 SIN(THETA2-BETA1+PI)
 END

END;

FOR J:=1 TO 3 DO
 VEC[J]:=VERTEX[J];
FOR J:=1 TO 2 DO
 GRAPHICSVERTEX[J]:=VEC[1]*GRAPHICSBASIS[J,1]+VEC[2]*
 GRAPHICSBASIS[J,2]+VEC[3]*GRAPHICSBASIS[J,3]
END;

IF INITIAL THEN
 WITH BOXPOLYHEDRON@ DO

  BEGIN
  WITH CURRPOLYVERTEX@ DO

   BEGIN
   VERTEXNUMBER:=I;
   POLYVERTEXSET:=POLYVERTEXSET+[VERTEXNUMBER];
   VERTEXPLANESET:=[];
   NEXTVERTEX:=NIL
   END;
```

```
    IF POLYHEDRONVERTEX=NIL THEN
     POLYHEDRONVERTEX:=CURRPOLYVERTEX;
    IF PREVPOLYVERTEX<>NIL THEN
     PREVPOLYVERTEX@.NEXTVERTEX:=CURRPOLYVERTEX;
    PREVPOLYVERTEX:=CURRPOLYVERTEX
    END

   ELSE
    CURRPOLYVERTEX:=CURRPOLYVERTEX@.NEXTVERTEX

  END;

 RESET(BOX);
 PLANERECORDS(BOXPOLYHEDRON,BOX,INITIAL)
 END

END;

PROCEDURE INPUTOBSTACLES;

CONST
 DELTA=0.0001;

VAR
 I:1..3;
 J:0..N;
 DISPLACEMENT:REAL;
 VEC:VECTOR;
 FIRSTVERTEX:BOOLEAN;

PROCEDURE CONVEXITYTEST;

VAR
 I:1..3;
 VEC:VECTOR;
 FIRST:BOOLEAN;

BEGIN
WITH CURRPOLYHEDRON@ DO

 BEGIN
 CURRENTPLANE:=FIRSTPLANE;

 WHILE CURRENTPLANE<>NIL DO

  BEGIN
  WITH CURRENTPLANE@ DO
   IF NUMOFVERTICES>=3 THEN

    BEGIN
    FIRST:=TRUE;
    CURRPOLYVERTEX:=POLYHEDRONVERTEX;
```

```
    WHILE CURRPOLYVERTEX<>NIL DO

     BEGIN
     WITH CURRPOLYVERTEX@ DO
      IF NOT (VERTEXNUMBER IN PLANEVERTEXSET) THEN

       BEGIN
       FOR I:=1 TO 3 DO
        VEC[I]:=VERTEX[I]-PLANEVERTEX@.VERTEX[I];
       DISPLACEMENT:=VEC[1]*OUTWARDNORMAL[1]+VEC[2]*
       OUTWARDNORMAL[2]+VEC[3]*OUTWARDNORMAL[3];

       IF FIRST THEN

        BEGIN
        FIRST:=FALSE;
        IF DISPLACEMENT>0 THEN
         FOR I:=1 TO 3 DO
          OUTWARDNORMAL[I]:=-OUTWARDNORMAL[I]
        END

       ELSE
        IF DISPLACEMENT>0 THEN

         BEGIN
         WORKSPACEERROR:=TRUE;
         WRITELN(ERRORS,'AT PLANE NUMBER':15,PLANENUMBER:4,
         ' OF OBSTACLE NUMBER':19,POLYHEDRONNUMBER:4);
         WRITELN(ERRORS,'CONVEXITY FAILS DUE TO':22,
         ' VERTEX NUMBER':14,VERTEXNUMBER:4,' *':2)
         END

       END;

     CURRPOLYVERTEX:=CURRPOLYVERTEX@.NEXTVERTEX
     END

    END;

  CURRENTPLANE:=CURRENTPLANE@.NEXTPLANE
  END

 END

END;

BEGIN
RESET(OBSTS);
NUMOFOBSTACLES:=0;
FIRSTOBSTACLE:=NIL;
PREVPOLYHEDRON:=NIL;

WHILE NOT EOF(OBSTS) DO
 IF NOT EOLN(OBSTS) THEN
```

```
BEGIN
READ(OBSTS,CODE);

CASE CODE OF

' ':;

'O':

 BEGIN
 NUMOFOBSTACLES:=NUMOFOBSTACLES+1;
 NEW(CURRPOLYHEDRON);

 WITH CURRPOLYHEDRON@ DO

  BEGIN
  POLYHEDRONTYPE:=OBSTACLE;
  POLYHEDRONNUMBER:=NUMOFOBSTACLES;
  POLYVERTEXSET:=[];
  POLYHEDRONVERTEX:=NIL;
  PREVPOLYVERTEX:=NIL;
  FIRSTVERTEX:=TRUE;
  POLYPLANESET:=[];
  NUMOFPLANES:=0;
  VISIBLEPLANESET:=[];
  FIRSTPLANE:=NIL;
  PREVIOUSPLANE:=NIL;
  NEXTPOLYHEDRON:=NIL
  END;

 IF FIRSTOBSTACLE=NIL THEN
  FIRSTOBSTACLE:=CURRPOLYHEDRON;
 IF PREVPOLYHEDRON<>NIL THEN
  PREVPOLYHEDRON@.NEXTPOLYHEDRON:=CURRPOLYHEDRON;
 PREVPOLYHEDRON:=CURRPOLYHEDRON
 END;

'V':
 WITH CURRPOLYHEDRON@ DO

  BEGIN
  NEW(CURRPOLYVERTEX);

  WITH CURRPOLYVERTEX@ DO

   BEGIN
   IF EOLN(OBSTS) THEN
    READLN(OBSTS);
   READ(OBSTS,VERTEXNUMBER);
   POLYVERTEXSET:=POLYVERTEXSET+[VERTEXNUMBER];

   FOR I:=1 TO 3 DO
```

```
       BEGIN
       IF EOLN(OBSTS) THEN
        READLN(OBSTS);
       READ(OBSTS,VERTEX[I]);
       VEC[I]:=VERTEX[I]
       END;

      FOR I:=1 TO 2 DO
       GRAPHICSVERTEX[I]:=VEC[1]*GRAPHICSBASIS[I,1]+
       VEC[2]*GRAPHICSBASIS[I,2]+VEC[3]*GRAPHICSBASIS[I,3];
      FINDBOUNDS(CURRPOLYHEDRON,FIRSTVERTEX);
      VERTEXPLANESET:=[];
      NEXTVERTEX:=NIL
      END;

     IF POLYHEDRONVERTEX=NIL THEN
      POLYHEDRONVERTEX:=CURRPOLYVERTEX;
     IF PREVPOLYVERTEX<>NIL THEN
      PREVPOLYVERTEX@.NEXTVERTEX:=CURRPOLYVERTEX;
     PREVPOLYVERTEX:=CURRPOLYVERTEX
     END;

   '$':

    BEGIN
    CLASSIFYBOUNDS(CURRPOLYHEDRON);
    PLANERECORDS(CURRPOLYHEDRON,OBSTS,TRUE)
    END

   END

   END

  ELSE
   READLN(OBSTS);

 WORKSPACEERROR:=FALSE;
 CURRPOLYHEDRON:=FIRSTOBSTACLE;

 WHILE CURRPOLYHEDRON<>NIL DO

  BEGIN
  WITH CURRPOLYHEDRON@ DO

   BEGIN
   IF NUMOFPLANES<4 THEN

    BEGIN
    WORKSPACEERROR:=TRUE;
    WRITELN(ERRORS,'OBSTACLE NUMBER':15,POLYHEDRONNUMBER:4,
            ' HAS LESS THAN FOUR PLANES *':28)
    END;

   CURRENTPLANE:=FIRSTPLANE;
```

```
WHILE CURRENTPLANE<>NIL DO

 BEGIN
 WITH CURRENTPLANE@ DO

  BEGIN
  IF NUMOFVERTICES<3 THEN

   BEGIN
   WORKSPACEERROR:=TRUE;
   WRITELN(ERRORS,'PLANE NUMBER':12,PLANENUMBER:4,
           ' OF OBSTACLE NUMBER':19,POLYHEDRONNUMBER:4,
           ' HAS LESS THAN THREE VERTICES *':31)
   END;

  J:=0;
  CURRPLANEVERTEX:=PLANEVERTEX;

  WHILE CURRPLANEVERTEX<>NIL DO

   BEGIN
   J:=J+1;

   IF J>3 THEN
    WITH CURRPLANEVERTEX@ DO

     BEGIN
     FOR I:=1 TO 3 DO
      VEC[I]:=VERTEX[I]-PLANEVERTEX@.VERTEX[I];
     DISPLACEMENT:=VEC[1]*OUTWARDNORMAL[1]+
     VEC[2]*OUTWARDNORMAL[2]+VEC[3]*OUTWARDNORMAL[3];
     IF ABS(DISPLACEMENT)>DELTA THEN

      BEGIN
      WORKSPACEERROR:=TRUE;
      WRITELN(ERRORS,'VERTEX NUMBER':13,VERTEXNUMBER:4,
              ' OF PLANE NUMBER':16,PLANENUMBER:4,
              ' OF OBSTACLE NUMBER':19,
              CURRPOLYHEDRON@.POLYHEDRONNUMBER:4);
      WRITELN(ERRORS,'DOES NOT LIE IN THE PLANE *':27)
      END

     END;

   CURRPLANEVERTEX:=CURRPLANEVERTEX@.NEXTPLANEVERTEX
   END

  END;

 CURRENTPLANE:=CURRENTPLANE@.NEXTPLANE
 END;

CONVEXITYTEST
```

```
  END;

 CURRPOLYHEDRON:=CURRPOLYHEDRON@.NEXTPOLYHEDRON
 END

END;

PROCEDURE INPUTWORKPIECES;

VAR
 I:1..3;
 ORIENTATION:REAL;

BEGIN
RESET(WKPCS);
WORKPIECEERROR:=FALSE;
FIRSTWORKPIECE:=NIL;
PREVWORKPIECE:=NIL;

WHILE NOT EOF(WKPCS) DO

 BEGIN
 NEW(CURRWORKPIECE);

 WITH CURRWORKPIECE@ DO

  BEGIN
  NEW(BOXPOLYHEDRON);

  WITH BOXPOLYHEDRON@ DO

   BEGIN
   POLYHEDRONTYPE:=FREEWORKPIECE;
   READLN(WKPCS,POLYHEDRONNUMBER,LENGTH,WIDTH,HEIGHT);

   IF LENGTH>WWRIST THEN

    BEGIN
    WORKPIECEERROR:=TRUE;
    WRITELN(ERRORS,'WORKPIECE NUMBER':16,POLYHEDRONNUMBER:4,
            ' IS TOO LONG *':14)
    END;

   IF WIDTH>(LWRIST-2*WFINGER-2*CLEARANCE-SAFETYFACTOR) THEN

    BEGIN
    WORKPIECEERROR:=TRUE;
    WRITELN(ERRORS,'WORKPIECE NUMBER':16,POLYHEDRONNUMBER:4,
            ' IS TOO WIDE *':14)
    END;

   IF HEIGHT>2*HFINGER THEN

    BEGIN
```

```
      WORKPIECEERROR:=TRUE;
      WRITELN(ERRORS,'WORKPIECE NUMBER':16,POLYHEDRONNUMBER:4,
              ' IS TOO HIGH *':14)
      END;

     IF HEIGHT<(HFINGER+SAFETYFACTOR) THEN

      BEGIN
      WORKPIECEERROR:=TRUE;
      WRITELN(ERRORS,'WORKPIECE NUMBER':16,POLYHEDRONNUMBER:4,
              ' IS NOT HIGH ENOUGH *':21)
      END;

    HYPOTENUSE:=0.5*SQRT(SQR(LENGTH)+SQR(WIDTH));
    BETA:=ARCTAN(WIDTH/LENGTH);
    FOR I:=1 TO 3 DO
     READ(WKPCS,REFERENCE[I]);
    READ(WKPCS,ORIENTATION);
    READLN(WKPCS);
    THETA:=ORIENTATION*PI/180;
    BOXRECORDS(CURRWORKPIECE,TRUE);
    NEXTDEFINEBOX:=NIL
    END

   END;

  IF FIRSTWORKPIECE=NIL THEN
   FIRSTWORKPIECE:=CURRWORKPIECE;
  IF PREVWORKPIECE<>NIL THEN
   PREVWORKPIECE@.NEXTDEFINEBOX:=CURRWORKPIECE;
  PREVWORKPIECE:=CURRWORKPIECE
  END

END;

PROCEDURE CONFIGURATION(VAR POINTER1:POINTMANIPULATOR;
                        TYPENAME:MANIPULATORTYPES;KVALUE:REAL;
                        POINTER2:POINTDEFINEBOX;INITIAL:BOOLEAN;
                        HOLDSTATUS:HOLDSTATUSNAMES);

VAR
 I:1..3;
 R1,R2,R3,GAMMA:REAL;

PROCEDURE WORKPIECERECORDS;

BEGIN
WITH POINTER1@ DO

 BEGIN
 NEW(WORKPIECE);

 WITH WORKPIECE@ DO
```

```
    BEGIN
    LENGTH:=POINTER2@.LENGTH;
    WIDTH:=POINTER2@.WIDTH;
    HEIGHT:=POINTER2@.HEIGHT;
    HYPOTENUSE:=POINTER2@.HYPOTENUSE;
    BETA:=POINTER2@.BETA;
    REFERENCE[1]:=WRIST@.REFERENCE[1];
    REFERENCE[2]:=WRIST@.REFERENCE[2];
    REFERENCE[3]:=WRIST@.REFERENCE[3]-CLEARANCE-HEIGHT;
    THETA:=ORIENTATION+PI/2;
    NEW(BOXPOLYHEDRON);
    BOXPOLYHEDRON@.POLYHEDRONTYPE:=HELDWORKPIECE;
    BOXRECORDS(WORKPIECE,TRUE)
    END

  END

 END;

 BEGIN
 IF INITIAL THEN

  BEGIN
  NEW(POINTER1);

  WITH POINTER1@ DO

   BEGIN
   MANIPULATORTYPE:=TYPENAME;
   K:=KVALUE;
   NEW(BASE); NEW(LINK1); NEW(LINK2); NEW(LINK3);
   NEW(WRIST); NEW(FINGER1); NEW(FINGER2)
   END

  END;

 WITH POINTER1@ DO

  BEGIN
  DOFANGLE:=ALPHA;
  ORIENTATION:=ANGLE;
  GAP:=DISTANCE;

  WITH BASE@ DO

   BEGIN
   IF INITIAL THEN

    BEGIN
    LENGTH:=LBASE;
    WIDTH:=THICKNESS;
    HEIGHT:=HBASE;
    HYPOTENUSE:=0.5*SQRT(SQR(LENGTH)+SQR(WIDTH));
    BETA:=ARCTAN(WIDTH/LENGTH);
```

```
  FOR I:=1 TO 3 DO
   REFERENCE[I]:=0;
  NEW(BOXPOLYHEDRON);
  BOXPOLYHEDRON@.POLYHEDRONTYPE:=BASENAME
  END;

 THETA:=DOFANGLE[1];
 BOXRECORDS(BASE,INITIAL)
 END;

WITH LINK1@ DO

 BEGIN
 IF INITIAL THEN

  BEGIN
  LENGTH:=LLINK1;
  WIDTH:=THICKNESS;
  HEIGHT1:=H1LINK1;
  HEIGHT2:=H2LINK1;
  HYPOTENUSE1:=0.5*SQRT(SQR(LENGTH)+SQR(HEIGHT1));
  HYPOTENUSE2:=0.5*SQRT(SQR(LENGTH)+SQR(HEIGHT2));
  BETA1:=ARCTAN(HEIGHT1/LENGTH);
  BETA2:=ARCTAN(HEIGHT2/LENGTH);
  NEW(BOXPOLYHEDRON);
  BOXPOLYHEDRON@.POLYHEDRONTYPE:=LINK1NAME
  END;

 R1:=THICKNESS+CLEARANCE;
 R2:=0.4*LLINK1*COS(PI/2-DOFANGLE[2]);
 R3:=SQRT(SQR(R1)+SQR(R2));
 IF R2>0 THEN
  GAMMA:=PI/2-ARCTAN(R2/R1)
 ELSE
  GAMMA:=PI/2+ARCTAN(-R2/R1);
 REFERENCE[1]:=R3*COS(DOFANGLE[1]+GAMMA);
 REFERENCE[2]:=R3*SIN(DOFANGLE[1]+GAMMA);
 REFERENCE[3]:=0.9*HBASE+0.4*LLINK1*SIN(PI/2-DOFANGLE[2]);
 THETA1:=DOFANGLE[1];
 THETA2:=PI/2-DOFANGLE[2];
 LINKRECORDS(LINK1,INITIAL)
 END;

WITH LINK2@ DO

 BEGIN
 IF INITIAL THEN

  BEGIN
  LENGTH:=LLINK2;
  WIDTH:=THICKNESS;
  HEIGHT1:=H1LINK2;
  HEIGHT2:=H2LINK2;
  HYPOTENUSE1:=0.5*SQRT(SQR(LENGTH)+SQR(HEIGHT1));
```

```
   HYPOTENUSE2:=0.5*SQRT(SQR(LENGTH)+SQR(HEIGHT2));
   BETA1:=ARCTAN(HEIGHT1/LENGTH);
   BETA2:=ARCTAN(HEIGHT2/LENGTH);
   NEW(BOXPOLYHEDRON);
   BOXPOLYHEDRON@.POLYHEDRONTYPE:=LINK2NAME
   END;

  R2:=0.8*LLINK1*COS(PI/2-DOFANGLE[2])+
  0.4*LLINK2*COS(PI/2-DOFANGLE[2]-DOFANGLE[3]);
  REFERENCE[1]:=R2*COS(DOFANGLE[1]);
  REFERENCE[2]:=R2*SIN(DOFANGLE[1]);
  REFERENCE[3]:=0.9*HBASE+0.8*LLINK1*SIN(PI/2-DOFANGLE[2])+
  0.4*LLINK2*SIN(PI/2-DOFANGLE[2]-DOFANGLE[3]);
  THETA1:=DOFANGLE[1];
  THETA2:=PI/2-DOFANGLE[2]-DOFANGLE[3];
  LINKRECORDS(LINK2,INITIAL)
  END;

 WITH LINK3@ DO

  BEGIN
  IF INITIAL THEN

   BEGIN
   LENGTH:=LLINK3;
   WIDTH:=THICKNESS;
   HEIGHT:=HLINK3;
   HYPOTENUSE:=0.5*SQRT(SQR(LENGTH)+SQR(WIDTH));
   BETA:=ARCTAN(WIDTH/LENGTH);
   NEW(BOXPOLYHEDRON);
   BOXPOLYHEDRON@.POLYHEDRONTYPE:=LINK3NAME
   END;

  R1:=THICKNESS+CLEARANCE;
  R2:=0.8*LLINK1*COS(PI/2-DOFANGLE[2])+
  0.8*LLINK2*COS(PI/2-DOFANGLE[2]-DOFANGLE[3]);
  R3:=SQRT(SQR(R1)+SQR(R2));
  IF R2>0 THEN
   GAMMA:=PI/2-ARCTAN(R2/R1)
  ELSE
   GAMMA:=PI/2+ARCTAN(-R2/R1);
  REFERENCE[1]:=R3*COS(DOFANGLE[1]+GAMMA);
  REFERENCE[2]:=R3*SIN(DOFANGLE[1]+GAMMA);
  REFERENCE[3]:=0.9*HBASE+0.8*LLINK1*SIN(PI/2-DOFANGLE[2])+
  0.8*LLINK2*SIN(PI/2-DOFANGLE[2]-DOFANGLE[3])-0.9*HLINK3;
  THETA:=DOFANGLE[1];
  BOXRECORDS(LINK3,INITIAL)
  END;

 WITH WRIST@ DO

  BEGIN
  IF INITIAL THEN
```

```
  BEGIN
  LENGTH:=LWRIST;
  WIDTH:=WWRIST;
  HEIGHT:=HWRIST;
  HYPOTENUSE:=0.5*SQRT(SQR(LENGTH)+SQR(WIDTH));
  BETA:=ARCTAN(WIDTH/LENGTH);
  NEW(BOXPOLYHEDRON);
  BOXPOLYHEDRON@.POLYHEDRONTYPE:=WRISTNAME
  END;

 REFERENCE[1]:=LINK3@.REFERENCE[1];
 REFERENCE[2]:=LINK3@.REFERENCE[2];
 REFERENCE[3]:=LINK3@.REFERENCE[3]-CLEARANCE-HEIGHT;
 THETA:=ORIENTATION;
 BOXRECORDS(WRIST,INITIAL)
 END;

WITH FINGER1@ DO

 BEGIN
 IF INITIAL THEN

  BEGIN
  LENGTH:=LFINGER;
  WIDTH:=WFINGER;
  HEIGHT:=HFINGER;
  HYPOTENUSE:=0.5*SQRT(SQR(LENGTH)+SQR(WIDTH));
  BETA:=ARCTAN(WIDTH/LENGTH);
  NEW(BOXPOLYHEDRON);
  BOXPOLYHEDRON@.POLYHEDRONTYPE:=FINGER1NAME
  END;

 REFERENCE[1]:=WRIST@.REFERENCE[1]+0.5*(WFINGER+GAP)*
 COS(ORIENTATION);
 REFERENCE[2]:=WRIST@.REFERENCE[2]+0.5*(WFINGER+GAP)*
 SIN(ORIENTATION);
 REFERENCE[3]:=WRIST@.REFERENCE[3]-CLEARANCE-HEIGHT;
 THETA:=ORIENTATION+PI/2;
 BOXRECORDS(FINGER1,INITIAL)
 END;

WITH FINGER2@ DO

 BEGIN
 IF INITIAL THEN

  BEGIN
  LENGTH:=LFINGER;
  WIDTH:=WFINGER;
  HEIGHT:=HFINGER;
  HYPOTENUSE:=0.5*SQRT(SQR(LENGTH)+SQR(WIDTH));
  BETA:=ARCTAN(WIDTH/LENGTH);
  NEW(BOXPOLYHEDRON);
  BOXPOLYHEDRON@.POLYHEDRONTYPE:=FINGER2NAME
```

```
  END;

 REFERENCE[1]:=WRIST@.REFERENCE[1]+0.5*(WFINGER+GAP)*
 COS(ORIENTATION-PI);
 REFERENCE[2]:=WRIST@.REFERENCE[2]+0.5*(WFINGER+GAP)*
 SIN(ORIENTATION-PI);
 REFERENCE[3]:=WRIST@.REFERENCE[3]-CLEARANCE-HEIGHT;
 THETA:=ORIENTATION+PI/2;
 BOXRECORDS(FINGER2,INITIAL)
 END;

CASE HOLDSTATUS OF

HOLDNEWWORKPIECE:
 IF INITIAL THEN
  WORKPIECERECORDS
 ELSE
  IF WORKPIECE=NIL THEN
   WORKPIECERECORDS
  ELSE
   WITH WORKPIECE@ DO

    BEGIN
    LENGTH:=POINTER2@.LENGTH;
    WIDTH:=POINTER2@.WIDTH;
    HEIGHT:=POINTER2@.HEIGHT;
    HYPOTENUSE:=POINTER2@.HYPOTENUSE;
    BETA:=POINTER2@.BETA;
    REFERENCE[1]:=WRIST@.REFERENCE[1];
    REFERENCE[2]:=WRIST@.REFERENCE[2];
    REFERENCE[3]:=WRIST@.REFERENCE[3]-CLEARANCE-HEIGHT;
    THETA:=ORIENTATION+PI/2;
    BOXRECORDS(WORKPIECE,FALSE)
    END;

HOLDOLDWORKPIECE:
 WITH WORKPIECE@ DO

  BEGIN
  REFERENCE[1]:=WRIST@.REFERENCE[1];
  REFERENCE[2]:=WRIST@.REFERENCE[2];
  REFERENCE[3]:=WRIST@.REFERENCE[3]-CLEARANCE-HEIGHT;
  THETA:=ORIENTATION+PI/2;
  BOXRECORDS(WORKPIECE,FALSE)
  END;

NOWORKPIECE:
 IF INITIAL THEN
  WORKPIECE:=NIL

END

END
```

```
END;

PROCEDURE FINDALPHA;

CONST
 DELTA=0.0001;

VAR
 L1,L2,R1,R2,R3,R4,R5,GAMMA1,GAMMA2:REAL;

FUNCTION ARCCOS(COSINE:REAL):REAL;

BEGIN
IF ABS(COSINE)<=DELTA THEN
 ARCCOS:=PI/2
ELSE
 IF COSINE>0 THEN
  ARCCOS:=ARCTAN(SQRT(1-SQR(COSINE))/COSINE)
 ELSE
  ARCCOS:=PI-ARCTAN(-SQRT(1-SQR(COSINE))/COSINE)
END;

BEGIN
L1:=0.8*LLINK1;
L2:=0.8*LLINK2;
R1:=THICKNESS+CLEARANCE;
R3:=SQRT(SQR(LOCATION[1])+SQR(LOCATION[2]));

IF R3<=R1 THEN

 BEGIN
 WRITELN(ERRORS,'TOO CLOSE TO BASE OF MANIPULATOR *':34);
 GOTO 1
 END;

R2:=SQRT(SQR(R3)-SQR(R1));
R4:=LOCATION[3]+HWRIST+CLEARANCE+0.9*(HLINK3-HBASE);
R5:=SQRT(SQR(R2)+SQR(R4));

IF R5<=ABS(L1-L2) THEN

 BEGIN
 WRITELN(ERRORS,'TOO CLOSE FOR MANIPULATOR TO REACH *':36);
 GOTO 1
 END;

IF R5>=(L1+L2) THEN

 BEGIN
 WRITELN(ERRORS,'TOO FAR FOR MANIPULATOR TO REACH *':34);
 GOTO 1
 END;

GAMMA1:=PI/2-ARCTAN(R2/R1);
```

```
ALPHA[1]:=ADJUSTANGLE(INVTAN(LOCATION[1],LOCATION[2])-GAMMA1);

IF ((95*PI/180)>=ALPHA[1]) AND (ALPHA[1]>=(85*PI/180)) THEN

 BEGIN
 WRITELN(ERRORS,'DOFANGLE[1] IS IN DEADBAND *':28);
 GOTO 1
 END;

IF R2<DELTA THEN

 BEGIN
 IF R4>0 THEN
  GAMMA1:=PI/2
 ELSE
  GAMMA1:=-PI/2
 END

ELSE
 IF R4>0 THEN
  GAMMA1:=ARCTAN(R4/R2)
 ELSE
  GAMMA1:=-ARCTAN(-R4/R2);

GAMMA2:=PI/2-ARCCOS((SQR(L1)+SQR(R5)-SQR(L2))/(2*L1*R5));
ALPHA[2]:=GAMMA2-GAMMA1;

IF (ALPHA[2]<(-80*PI/180)) OR (ALPHA[2]>(80*PI/180)) THEN

 BEGIN
 WRITELN(ERRORS,'DOFANGLE[2] IS OUT OF RANGE *':29);
 GOTO 1
 END;

ALPHA[3]:=PI-ARCCOS((SQR(L1)+SQR(L2)-SQR(R5))/(2*L1*L2));

IF ALPHA[3]>(135*PI/180) THEN

 BEGIN
 WRITELN(ERRORS,'DOFANGLE[3] IS OUT OF RANGE *':29);
 GOTO 1
 END

END;

PROCEDURE PRINTINTEGER(I:INTEGER;X,Y,SIZE:REAL);

VAR
 J,K:INTEGER;

BEGIN
K:=0;

WHILE I<>0 DO
```

```
 BEGIN
 J:=I MOD 10;
 I:=I DIV 10;

 CASE J OF
 0:MESSAGE:='0';
 1:MESSAGE:='1';
 2:MESSAGE:='2';
 3:MESSAGE:='3';
 4:MESSAGE:='4';
 5:MESSAGE:='5';
 6:MESSAGE:='6';
 7:MESSAGE:='7';
 8:MESSAGE:='8';
 9:MESSAGE:='9'
 END;

 SYMBOL(X-K*SIZE,Y,SIZE,MESSAGE,0,1);
 K:=K+1
 END

END;

PROCEDURE PRINTPAGENUMBER;

BEGIN
IF PAGENUMBER>99 THEN

 BEGIN
 MESSAGE:='-      -';
 SYMBOL(4.15,0.85,0.1,MESSAGE,0,7);
 PRINTINTEGER(PAGENUMBER,4.55,0.85,0.1)
 END

ELSE
 IF PAGENUMBER>9 THEN

  BEGIN
  MESSAGE:='-     -';
  SYMBOL(4.2,0.85,0.1,MESSAGE,0,6);
  PRINTINTEGER(PAGENUMBER,4.5,0.85,0.1)
  END

 ELSE

  BEGIN
  MESSAGE:='-    -';
  SYMBOL(4.25,0.85,0.1,MESSAGE,0,5);
  PRINTINTEGER(PAGENUMBER,4.45,0.85,0.1)
  END;

PAGENUMBER:=PAGENUMBER+1
END;
```

```
PROCEDURE CHECKPATH(HOLDSTATUS:HOLDSTATUSNAMES);

CONST
 LIMIT=69;
 STEPSIZE=2;

VAR
 I:2..3;
 CALLNUMBER:1..3;
 ITERATION:0..LIMIT;
 J:INTEGER;
 M,B,GAMMA:REAL;
 ADJUSTED,SAFEFLAG1,SAFEFLAG2,SAFEENDS:BOOLEAN;
 OBSTACLESET,WORKPIECESET:NUMBERSET;

PROCEDURE FINDPOLYPAIRS(POLYHEDRONA:POINTPOLYHEDRON;
                        VAR SAFEPOLYHEDRONA:BOOLEAN);

PROCEDURE CHECKSAFENESS(POLYHEDRONB:POINTPOLYHEDRON);

CONST
 DELTA=5;

TYPE
 SAFENESSTYPES=(SAFE,INDETERMINANT,UNSAFE);

VAR
 EXIT:BOOLEAN;
 PLANESETA,PLANESETB:NUMBERSET;
 VERTEXI,VERTEXII:POINTVERTEX;
 PLANEI:POINTPLANE;

PROCEDURE GENERATESETS(POLYHEDRONI,POLYHEDRONII:POINTPOLYHEDRON);

VAR
 FIRST:BOOLEAN;

BEGIN
PLANEI:=POLYHEDRONI@.FIRSTPLANE;
FIRST:=TRUE;

WHILE PLANEI<>NIL DO

 BEGIN
 WITH PLANEI@ DO

  BEGIN
  PDVERTEXSET:=[];
  VERTEXII:=POLYHEDRONII@.POLYHEDRONVERTEX;

  WHILE VERTEXII<>NIL DO

   BEGIN
```

```
   WITH VERTEXII@ DO

     BEGIN
     IF FIRST THEN
      NDPLANESET:=[];
     IF (OUTWARDNORMAL[1]*(VERTEX[1]-PLANEVERTEX@.VERTEX[1])+
     OUTWARDNORMAL[2]*(VERTEX[2]-PLANEVERTEX@.VERTEX[2])+
     OUTWARDNORMAL[3]*(VERTEX[3]-PLANEVERTEX@.VERTEX[3]))
     >DELTA THEN
      PDVERTEXSET:=PDVERTEXSET+[VERTEXNUMBER]
     ELSE
      NDPLANESET:=NDPLANESET+[PLANENUMBER]
     END;

   VERTEXII:=VERTEXII@.NEXTVERTEX
   END

  END;

 FIRST:=FALSE;
 PLANEI:=PLANEI@.NEXTPLANE
 END

END;

FUNCTION EXAMINESETSI(POLYHEDRONI,POLYHEDRONII:POINTPOLYHEDRON)
                       :SAFENESSTYPES;

BEGIN
EXAMINESETSI:=INDETERMINANT;
PLANEI:=POLYHEDRONI@.FIRSTPLANE;
EXIT:=FALSE;

WHILE (PLANEI<>NIL) AND NOT EXIT DO
 IF PLANEI@.PDVERTEXSET=POLYHEDRONII@.POLYVERTEXSET THEN

  BEGIN
  EXAMINESETSI:=SAFE;
  EXIT:=TRUE
  END

 ELSE
  PLANEI:=PLANEI@.NEXTPLANE;

IF NOT EXIT THEN

 BEGIN
 VERTEXII:=POLYHEDRONII@.POLYHEDRONVERTEX;

 WHILE (VERTEXII<>NIL) AND NOT EXIT DO
  IF VERTEXII@.NDPLANESET=POLYHEDRONI@.POLYPLANESET THEN

   BEGIN
   EXAMINESETSI:=UNSAFE;
```

```
    EXIT:=TRUE
    END

   ELSE
    VERTEXII:=VERTEXII@.NEXTVERTEX
  END

END;

FUNCTION EXAMINESETSII(POLYHEDRONI,POLYHEDRONII:POINTPOLYHEDRON)
                       :SAFENESSTYPES;

VAR
 TOTALPDVERTEXSET,PLANESET:NUMBERSET;

BEGIN
EXAMINESETSII:=UNSAFE;
VERTEXI:=POLYHEDRONI@.POLYHEDRONVERTEX;
EXIT:=FALSE;

WHILE (VERTEXI<>NIL) AND NOT EXIT DO

 BEGIN
 PLANEI:=POLYHEDRONI@.FIRSTPLANE;
 TOTALPDVERTEXSET:=[];
 PLANESET:=[];

 WHILE PLANEI<>NIL DO

  BEGIN
  IF PLANEI@.PLANENUMBER IN VERTEXI@.VERTEXPLANESET THEN

   BEGIN
   TOTALPDVERTEXSET:=TOTALPDVERTEXSET+PLANEI@.PDVERTEXSET;
   PLANESET:=PLANESET+[PLANEI@.PLANENUMBER];
   END;

  IF PLANESET=VERTEXI@.VERTEXPLANESET THEN

   BEGIN
   IF TOTALPDVERTEXSET=POLYHEDRONII@.POLYVERTEXSET THEN

    BEGIN
    EXAMINESETSII:=INDETERMINANT;
    EXIT:=TRUE
    END;

   PLANEI:=NIL
   END

  ELSE
   PLANEI:=PLANEI@.NEXTPLANE
  END;
```

```
 VERTEXI:=VERTEXI@.NEXTVERTEX
 END

END;

FUNCTION EXAMINESETSIII(POLYHEDRONI,POLYHEDRONII:POINTPOLYHEDRON;
                        VAR PLANESETI:NUMBERSET):SAFENESSTYPES;

VAR
 TOTALNDPLANESET:NUMBERSET;
 VERTEXI:POINTPLANEVERTEX;

BEGIN
PLANESETI:=[];
PLANEI:=POLYHEDRONI@.FIRSTPLANE;

WHILE PLANEI<>NIL DO

 BEGIN
 VERTEXI:=PLANEI@.PLANEVERTEX;
 TOTALNDPLANESET:=[];

 WHILE VERTEXI<>NIL DO

  BEGIN
  CURRPOLYVERTEX:=POLYHEDRONI@.POLYHEDRONVERTEX;
  WHILE CURRPOLYVERTEX@.VERTEXNUMBER<>VERTEXI@.VERTEXNUMBER DO
   CURRPOLYVERTEX:=CURRPOLYVERTEX@.NEXTVERTEX;
  TOTALNDPLANESET:=TOTALNDPLANESET+CURRPOLYVERTEX@.NDPLANESET;
  VERTEXI:=VERTEXI@.NEXTPLANEVERTEX
  END;

 IF TOTALNDPLANESET=POLYHEDRONII@.POLYVERTEXSET THEN
  PLANESETI:=PLANESETI+[PLANEI@.PLANENUMBER];
 PLANEI:=PLANEI@.NEXTPLANE
 END;

IF PLANESETI=[] THEN
 EXAMINESETSIII:=SAFE
ELSE
 EXAMINESETSIII:=INDETERMINANT
END;

FUNCTION EXAMINESETSIV:SAFENESSTYPES;

VAR
 VERTEXSETA,VERTEXSETB:NUMBERSET;
 VERTEXUA,VERTEXVA,VERTEXUB,VERTEXVB,
 VERTEXDA,VERTEXDB:POINTVERTEX;
 PLANESA,PLANETA,PLANESB,PLANETB:POINTPLANE;

FUNCTION FOUNDEDGE(POLYHEDRONI,POLYHEDRONII:POINTPOLYHEDRON;
                   PLANESI,PLANETI:POINTPLANE;VERTEXSETI:NUMBERSET;
                   VAR VERTEXUI,VERTEXVI,VERTEXDI:POINTVERTEX):BOOLEAN;
```

```
BEGIN
VERTEXUI:=NIL;
VERTEXVI:=NIL;
CURRPOLYVERTEX:=POLYHEDRONI@.POLYHEDRONVERTEX;

WHILE CURRPOLYVERTEX<>NIL DO
 IF CURRPOLYVERTEX@.VERTEXNUMBER IN VERTEXSETI THEN

  BEGIN
  IF VERTEXUI=NIL THEN

   BEGIN
   VERTEXUI:=CURRPOLYVERTEX;
   CURRPOLYVERTEX:=CURRPOLYVERTEX@.NEXTVERTEX
   END

  ELSE

   BEGIN
   VERTEXVI:=CURRPOLYVERTEX;
   CURRPOLYVERTEX:=NIL
   END

  END

 ELSE
  CURRPOLYVERTEX:=CURRPOLYVERTEX@.NEXTVERTEX;

IF VERTEXVI<>NIL THEN

 BEGIN
 IF (VERTEXUI@.NDPLANESET+VERTEXVI@.NDPLANESET)=
 POLYHEDRONII@.POLYPLANESET THEN

  BEGIN
  VERTEXDI:=POLYHEDRONA@.POLYHEDRONVERTEX;
  WHILE (VERTEXDI@.VERTEXNUMBER IN PLANESI@.PLANEVERTEXSET)
  OR (VERTEXDI@.VERTEXNUMBER IN PLANETI@.PLANEVERTEXSET) DO
   VERTEXDI:=VERTEXDI@.NEXTVERTEX;
  FOUNDEDGE:=TRUE
  END

 ELSE
  FOUNDEDGE:=FALSE
 END

ELSE
 FOUNDEDGE:=FALSE
END;

FUNCTION OUTOFREGIONIII(VERTEXUI,VERTEXVI,VERTEXDI,
                        VERTEXUII,VERTEXVII:POINTVERTEX):BOOLEAN;
```

```
VAR
 I:1..3;
 MAGNITUDE:REAL;
 VEC1,VEC2,VEC3,VEC4,NORMAL:VECTOR;

BEGIN
FOR I:=1 TO 3 DO

 BEGIN
 VEC1[I]:=VERTEXDI@.VERTEX[I]-VERTEXUI@.VERTEX[I];
 VEC2[I]:=VERTEXVI@.VERTEX[I]-VERTEXUI@.VERTEX[I];
 VEC3[I]:=VERTEXUII@.VERTEX[I]-VERTEXUI@.VERTEX[I];
 VEC4[I]:=VERTEXVII@.VERTEX[I]-VERTEXUI@.VERTEX[I]
 END;

NORMAL[1]:=VEC2[2]*VEC3[3]-VEC3[2]*VEC2[3];
NORMAL[2]:=VEC3[1]*VEC2[3]-VEC2[1]*VEC3[3];
NORMAL[3]:=VEC2[1]*VEC3[2]-VEC3[1]*VEC2[2];
MAGNITUDE:=SQRT(SQR(NORMAL[1])+SQR(NORMAL[2])
+SQR(NORMAL[3]));
FOR I:=1 TO 3 DO
 NORMAL[I]:=NORMAL[I]/MAGNITUDE;
IF (NORMAL[1]*VEC1[1]+NORMAL[2]*VEC1[2]+
NORMAL[3]*VEC1[3])>0 THEN
 FOR I:=1 TO 3 DO
  NORMAL[I]:=-NORMAL[I];

IF (NORMAL[1]*VEC4[1]+NORMAL[2]*VEC4[2]+
NORMAL[3]*VEC4[3])>DELTA THEN
 OUTOFREGIONIII:=TRUE
ELSE
 OUTOFREGIONIII:=FALSE
END;

BEGIN
EXAMINESETSIV:=UNSAFE;
EXIT:=FALSE;
PLANESA:=POLYHEDRONA@.FIRSTPLANE;

WHILE (PLANESA<>NIL) AND NOT EXIT DO

 BEGIN
 IF PLANESA@.PLANENUMBER IN PLANESETA THEN

  BEGIN
  PLANESETA:=PLANESETA-[PLANESA@.PLANENUMBER];
  PLANETA:=POLYHEDRONA@.FIRSTPLANE;

  WHILE (PLANETA<>NIL) AND NOT EXIT DO

   BEGIN
   IF PLANETA@.PLANENUMBER IN PLANESETA THEN

    BEGIN
```

```
VERTEXSETA:=PLANESA@.PLANEVERTEXSET*PLANETA@.PLANEVERTEXSET;

IF VERTEXSETA<>[] THEN

 BEGIN
 IF FOUNDEDGE(POLYHEDRONA,POLYHEDRONB,PLANESA,PLANETA,
 VERTEXSETA,VERTEXUA,VERTEXVA,VERTEXDA) THEN

  BEGIN
  PLANESB:=POLYHEDRONB@.FIRSTPLANE;

  WHILE (PLANESB<>NIL) AND NOT EXIT DO

   BEGIN
   IF PLANESB@.PLANENUMBER IN PLANESETB THEN

    BEGIN
    PLANESETB:=PLANESETB-[PLANESB@.PLANENUMBER];
    PLANETB:=POLYHEDRONB@.FIRSTPLANE;

    WHILE (PLANETB<>NIL) AND NOT EXIT DO

     BEGIN
     IF PLANETB@.PLANENUMBER IN PLANESETB THEN

      BEGIN
      VERTEXSETB:=PLANESB@.PLANEVERTEXSET*PLANETB@.PLANEVERTEXSET;

      IF VERTEXSETB<>[] THEN

       BEGIN
       IF FOUNDEDGE(POLYHEDRONB,POLYHEDRONA,PLANESB,PLANETB,
       VERTEXSETB,VERTEXUB,VERTEXVB,VERTEXDB) THEN

        BEGIN
        IF ((VERTEXUA@.VERTEXNUMBER IN PLANESB@.PDVERTEXSET)
        AND NOT (VERTEXUA@.VERTEXNUMBER IN PLANETB@.PDVERTEXSET)
        AND NOT (VERTEXVA@.VERTEXNUMBER IN PLANESB@.PDVERTEXSET)
        AND (VERTEXVA@.VERTEXNUMBER IN PLANETB@.PDVERTEXSET))
        OR (NOT (VERTEXUA@.VERTEXNUMBER IN PLANESB@.PDVERTEXSET)
        AND (VERTEXUA@.VERTEXNUMBER IN PLANETB@.PDVERTEXSET)
        AND (VERTEXVA@.VERTEXNUMBER IN PLANESB@.PDVERTEXSET)
        AND NOT (VERTEXVA@.VERTEXNUMBER IN PLANETB@.PDVERTEXSET))
        AND ((VERTEXUB@.VERTEXNUMBER IN PLANESA@.PDVERTEXSET)
        AND NOT (VERTEXUB@.VERTEXNUMBER IN PLANETA@.PDVERTEXSET)
        AND NOT (VERTEXVB@.VERTEXNUMBER IN PLANESA@.PDVERTEXSET)
        AND (VERTEXVB@.VERTEXNUMBER IN PLANETA@.PDVERTEXSET))
        OR (NOT (VERTEXUB@.VERTEXNUMBER IN PLANESA@.PDVERTEXSET)
        AND (VERTEXUB@.VERTEXNUMBER IN PLANETA@.PDVERTEXSET)
        AND (VERTEXVB@.VERTEXNUMBER IN PLANESA@.PDVERTEXSET)
        AND NOT (VERTEXVB@.VERTEXNUMBER IN PLANETA@.PDVERTEXSET))
        THEN

         BEGIN
```

```
             IF OUTOFREGIONIII(VERTEXUA,VERTEXVA,VERTEXDA,
             VERTEXUB,VERTEXVB)
             AND OUTOFREGIONIII(VERTEXUB,VERTEXVB,VERTEXDB,
             VERTEXUA,VERTEXVA) THEN

              BEGIN
              EXAMINESETSIV:=SAFE;
              EXIT:=TRUE
              END

             END

            END

           END

          END

         END;

        PLANETB:=PLANETB@.NEXTPLANE
        END

       END;

      PLANESB:=PLANESB@.NEXTPLANE
      END

     END

    END;

   PLANETA:=PLANETA@.NEXTPLANE
   END

  END;

 PLANESA:=PLANESA@.NEXTPLANE
 END

END;

BEGIN
GENERATESETS(POLYHEDRONA,POLYHEDRONB);

CASE EXAMINESETSI(POLYHEDRONA,POLYHEDRONB) OF
SAFE:;
INDETERMINANT:

 BEGIN
 GENERATESETS(POLYHEDRONB,POLYHEDRONA);

 CASE EXAMINESETSI(POLYHEDRONB,POLYHEDRONA) OF
 SAFE:;
```

```
 INDETERMINANT:

  CASE EXAMINESETSII(POLYHEDRONA,POLYHEDRONB) OF
  SAFE:;
  INDETERMINANT:

   CASE EXAMINESETSII(POLYHEDRONB,POLYHEDRONA) OF
   SAFE:;
   INDETERMINANT:

    CASE EXAMINESETSIII(POLYHEDRONA,POLYHEDRONB,PLANESETA) OF
    SAFE:;
    INDETERMINANT:

     CASE EXAMINESETSIII(POLYHEDRONB,POLYHEDRONA,PLANESETB) OF
     SAFE:;
     INDETERMINANT:

      CASE EXAMINESETSIV OF
      SAFE:;
      INDETERMINANT:;
      UNSAFE:SAFEPOLYHEDRONA:=FALSE
      END;

     UNSAFE:
     END;

    UNSAFE:
    END;

   UNSAFE:SAFEPOLYHEDRONA:=FALSE
   END;

  UNSAFE:SAFEPOLYHEDRONA:=FALSE
  END;

 UNSAFE:SAFEPOLYHEDRONA:=FALSE
 END

 END;

UNSAFE:SAFEPOLYHEDRONA:=FALSE
END

END;

BEGIN
CURRPOLYHEDRON:=FIRSTOBSTACLE;

WHILE CURRPOLYHEDRON<>NIL DO

 BEGIN
 IF CURRPOLYHEDRON@.POLYHEDRONNUMBER IN OBSTACLESET THEN
  CHECKSAFENESS(CURRPOLYHEDRON);
```

```
 CURRPOLYHEDRON:=CURRPOLYHEDRON@.NEXTPOLYHEDRON
 END;

CURRWORKPIECE:=FIRSTWORKPIECE;

WHILE CURRWORKPIECE<>NIL DO

 BEGIN
 WITH CURRWORKPIECE@ DO
  IF BOXPOLYHEDRON@.POLYHEDRONNUMBER IN WORKPIECESET THEN
   CHECKSAFENESS(BOXPOLYHEDRON);
 CURRWORKPIECE:=CURRWORKPIECE@.NEXTDEFINEBOX
 END

END;

PROCEDURE PLOTGRAPHS;

CONST
 XORGIN=1.5;

VAR
 YORGIN:REAL;

PROCEDURE PRINTSYMBOL(X,Y,SIZE:REAL;SYMBOLCODE,PLACECODE:INTEGER);

PROCEDURE SYMBOL(CONST X1,X2,X3:SHORTREAL;CONST X4:INTEGER;
                 CONST X5:SHORTREAL;CONST X6:INTEGER);FORTRAN;
BEGIN
SYMBOL(X,Y,SIZE,SYMBOLCODE,0,PLACECODE)
END;

PROCEDURE PLOTPOINT(X,Y:REAL);

VAR
 CODE:INTEGER;

BEGIN
IF ITERATION=0 THEN
 CODE:=2
ELSE
 IF CURRENTPATHSTEP@.SAFEMANIPULATOR THEN
  CODE:=0
 ELSE
  CODE:=11;
PRINTSYMBOL(X,Y,0.05,CODE,-1)
END;

BEGIN
CASE CALLNUMBER OF
1:YORGIN:=6.975;
2:YORGIN:=4.025;
3:YORGIN:=1.075
END;
```

```
PLOT(XORGIN,YORGIN,-3);

PLOT(0,2.95,2);
PLOT(6,2.95,2);
PLOT(6,0,2);
PLOT(0,0,2);

PLOT(0.4,0.65,3);
PLOT(0.4,2.45,2);
PLOT(0.4,1.55,3);
PLOT(2.6,1.55,2);
PLOT(0.425,0.75,3);
PLOT(0.375,0.75,2);
PLOT(0.425,2.35,3);
PLOT(0.375,2.35,2);
PLOT(0.5,1.525,3);
PLOT(0.5,1.575,2);
PLOT(2.5,1.525,3);
PLOT(2.5,1.575,2);

PLOT(3.4,0.9,3);
PLOT(3.4,2.45,2);
PLOT(3.4,0.9,3);
PLOT(5.6,0.9,2);
PLOT(3.425,1,3);
PLOT(3.375,1,2);
PLOT(3.425,2.35,3);
PLOT(3.375,2.35,2);
PLOT(3.5,0.875,3);
PLOT(3.5,0.925,2);
PLOT(5.5,0.875,3);
PLOT(5.5,0.925,2);

IF ITERATION=0 THEN

 BEGIN
 MESSAGE:='PATH';
 SYMBOL(1.9,2.7,0.1,MESSAGE,0,4);
 PRINTINTEGER(PATHNUMBER,2.5,2.7,0.1);
 MESSAGE:='LINEAR PATH';
 SYMBOL(3,2.7,0.1,MESSAGE,0,11);
 PRINTSYMBOL(2.51,0.325,0.05,2,-1);
 MESSAGE:='INDETERMINANT';
 SYMBOL(2.685,0.29,0.07,MESSAGE,0,13)
 END

ELSE

 BEGIN
 MESSAGE:='PATH';
 SYMBOL(1.85,2.7,0.1,MESSAGE,0,4);
 PRINTINTEGER(PATHNUMBER,2.45,2.7,0.1);
 MESSAGE:='ITERATION';
```

```
 SYMBOL(2.95,2.7,0.1,MESSAGE,0,9);
 PRINTINTEGER(ITERATION,4.05,2.7,0.1);
 PRINTSYMBOL(2.51,0.465,0.05,0,-1);
 MESSAGE:='SAFE';
 SYMBOL(2.685,0.43,0.07,MESSAGE,0,4);
 PRINTSYMBOL(2.51,0.325,0.05,11,-1);
 MESSAGE:='UNSAFE';
 SYMBOL(2.685,0.29,0.07,MESSAGE,0,6)
 END;

MESSAGE:='D.O.F.  DEGREE OF FREEDOM';
SYMBOL(2.125,0.15,0.07,MESSAGE,0,25);
MESSAGE:='=';
SYMBOL(2.58,0.15,0.07,MESSAGE,0,1);
MESSAGE:='-80';
SYMBOL(0.13,0.715,0.07,MESSAGE,0,3);
MESSAGE:='80';
SYMBOL(0.2,2.315,0.07,MESSAGE,0,2);
MESSAGE:='D.O.F. ANGLE 2';
SYMBOL(0.235,1.06,0.07,MESSAGE,90,14);
MESSAGE:='(DEGREES)';
SYMBOL(0.375,1.235,0.07,MESSAGE,90,9);
MESSAGE:='K';
SYMBOL(2.635,1.515,0.07,MESSAGE,0,1);
SYMBOL(5.635,0.865,0.07,MESSAGE,0,1);
MESSAGE:='0<K<1';
SYMBOL(2.65,1.36,0.07,MESSAGE,0,5);

MESSAGE:='0';
SYMBOL(3.27,0.965,0.07,MESSAGE,0,1);
SYMBOL(3.465,0.77,0.07,MESSAGE,0,1);
MESSAGE:='135';
SYMBOL(3.13,2.315,0.07,MESSAGE,0,3);
MESSAGE:='1';
SYMBOL(5.465,0.77,0.07,MESSAGE,0,1);
MESSAGE:='D.O.F. ANGLE 3';
SYMBOL(3.235,1.185,0.07,MESSAGE,90,14);
MESSAGE:='(DEGREES)';
SYMBOL(3.375,1.36,0.07,MESSAGE,90,9);

CURRENTPATHSTEP:=PATHSTART;

WHILE CURRENTPATHSTEP<>NIL DO

 BEGIN
 WITH CURRENTPATHSTEP@ DO
  CASE MANIPULATORTYPE OF
  FIXED:;

  VARIABLE:

   BEGIN
   PLOT(0.5,1.55,-3);
   PLOTPOINT(K*2,DOFANGLE[2]*1.8/PI);
```

```
    PLOT(-0.5,-1.55,-3);
    PLOT(3.5,1,-3);
    PLOTPOINT(K*2,DOFANGLE[3]*1.8/PI);
    PLOT(-3.5,-1,-3)
    END

   END;

  CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR
  END;

 PLOT(-XORGIN,-YORGIN,-3);

 CASE CALLNUMBER OF

 1,2:

  BEGIN
  CALLNUMBER:=CALLNUMBER+1;

  IF SAFEPATH OR (ITERATION=LIMIT) THEN

   BEGIN
   PRINTPAGENUMBER;
   PLOT(0,0,999)
   END

  END;

 3:

  BEGIN
  CALLNUMBER:=1;
  PRINTPAGENUMBER;
  PLOT(0,0,999)
  END

 END

 END;

 BEGIN
 SAFEENDS:=TRUE;
 SAFEPATH:=FALSE;
 ITERATION:=0;
 CALLNUMBER:=1;
 PLOTGRAPHS;

 WHILE NOT SAFEPATH AND (ITERATION<LIMIT) DO

  BEGIN
  ITERATION:=ITERATION+1;
  SAFEPATH:=TRUE;
  J:=1;
```

```
CURRENTPATHSTEP:=PATHSTART;

WHILE CURRENTPATHSTEP<>NIL DO

 BEGIN
 OBSTACLESET:=CURRENTPATHSTEP@.OBSTACLESET;
 WORKPIECESET:=CURRENTPATHSTEP@.WORKPIECESET;

 WITH CURRENTPATHSTEP@ DO
  CASE MANIPULATORTYPE OF

  FIXED:
   IF ITERATION=1 THEN

    BEGIN
    SAFEMANIPULATOR:=TRUE;
    FINDPOLYPAIRS(LINK1@.BOXPOLYHEDRON,SAFEMANIPULATOR);
    FINDPOLYPAIRS(LINK2@.BOXPOLYHEDRON,SAFEMANIPULATOR);
    FINDPOLYPAIRS(LINK3@.BOXPOLYHEDRON,SAFEMANIPULATOR);
    FINDPOLYPAIRS(WRIST@.BOXPOLYHEDRON,SAFEMANIPULATOR);
    FINDPOLYPAIRS(FINGER1@.BOXPOLYHEDRON,SAFEMANIPULATOR);
    FINDPOLYPAIRS(FINGER2@.BOXPOLYHEDRON,SAFEMANIPULATOR);

    CASE HOLDSTATUS OF
    HOLDNEWWORKPIECE,HOLDOLDWORKPIECE:
     FINDPOLYPAIRS(WORKPIECE@.BOXPOLYHEDRON,SAFEMANIPULATOR);
    NOWORKPIECE:
    END;

    IF NOT SAFEMANIPULATOR THEN

     BEGIN
     SAFEENDS:=FALSE;
     WRITELN(ERRORS,'PATH':4,PATHNUMBER:4,' STEP':5,J:4,
             ' FIXED CONFIGURATION UNSAFE *':29)
     END

    END;

  VARIABLE:

   BEGIN
   ADJUSTED:=FALSE;

   FOR I:=2 TO 3 DO

    BEGIN
    M:=(NEXTMANIPULATOR@.DOFANGLE[I]-PREVIOUSPATHSTEP@.DOFANGLE[I])
    /(NEXTMANIPULATOR@.K-PREVIOUSPATHSTEP@.K);
    B:=PREVIOUSPATHSTEP@.DOFANGLE[I]-M*PREVIOUSPATHSTEP@.K;
    GAMMA:=M*K+B;
    IF ((GAMMA>DOFANGLE[I]) AND (DOFANGLE[I]<0))
    OR ((GAMMA<DOFANGLE[I]) AND (DOFANGLE[I]>0)) THEN
```

```
  BEGIN
  DOFANGLE[I]:=GAMMA;
  ADJUSTED:=TRUE
  END

 END;

IF ADJUSTED THEN

 BEGIN
 ALPHA:=DOFANGLE;
 ANGLE:=ORIENTATION;
 DISTANCE:=GAP;

 CASE HOLDSTATUS OF
 HOLDNEWWORKPIECE,HOLDOLDWORKPIECE:
  CONFIGURATION(CURRENTPATHSTEP,VARIABLE,K,WKPCPOINTER,FALSE,
                HOLDOLDWORKPIECE);
 NOWORKPIECE:
  CONFIGURATION(CURRENTPATHSTEP,VARIABLE,K,NIL,FALSE,NOWORKPIECE)
 END

 END;

IF (ITERATION=1) OR ADJUSTED THEN
 SAFEMANIPULATOR:=FALSE;

IF NOT SAFEMANIPULATOR THEN

 BEGIN
 SAFEFLAG1:=TRUE;
 FINDPOLYPAIRS(LINK1@.BOXPOLYHEDRON,SAFEFLAG1);
 SAFEFLAG2:=TRUE;
 FINDPOLYPAIRS(LINK2@.BOXPOLYHEDRON,SAFEFLAG2);
 IF SAFEFLAG2 THEN
  FINDPOLYPAIRS(LINK3@.BOXPOLYHEDRON,SAFEFLAG2);
 IF SAFEFLAG2 THEN
  FINDPOLYPAIRS(WRIST@.BOXPOLYHEDRON,SAFEFLAG2);
 IF SAFEFLAG2 THEN
  FINDPOLYPAIRS(FINGER1@.BOXPOLYHEDRON,SAFEFLAG2);
 IF SAFEFLAG2 THEN
  FINDPOLYPAIRS(FINGER2@.BOXPOLYHEDRON,SAFEFLAG2);

 IF SAFEFLAG2 THEN
  CASE HOLDSTATUS OF
  HOLDNEWWORKPIECE,HOLDOLDWORKPIECE:

   BEGIN
   FINDPOLYPAIRS(WORKPIECE@.BOXPOLYHEDRON,SAFEFLAG2);
   IF WORKPIECE@.BOXPOLYHEDRON@.POLYHEDRONVERTEX@.VERTEX[3]
   <(0.5*LIFTHEIGHT) THEN
    SAFEFLAG2:=FALSE
   END;
```

```
  NOWORKPIECE:
   IF FINGER1@.BOXPOLYHEDRON@.POLYHEDRONVERTEX@.
   VERTEX[3]<(0.5*LIFTHEIGHT) THEN
    SAFEFLAG2:=FALSE
  END;

 IF SAFEFLAG1 AND SAFEFLAG2 THEN
  SAFEMANIPULATOR:=TRUE
 ELSE

  BEGIN
  SAFEMANIPULATOR:=FALSE;
  SAFEPATH:=FALSE;

  IF ITERATION=LIMIT-1 THEN
   WRITELN(ERRORS,'PATH':4,PATHNUMBER:4,' STEP':5,J:4,
           ' IS UNSAFE *':12)
  ELSE

   BEGIN
   IF DOFANGLE[2]>0 THEN
    DOFANGLE[2]:=DOFANGLE[2]-STEPSIZE*PI/180
   ELSE
    DOFANGLE[2]:=DOFANGLE[2]+STEPSIZE*PI/180;

   IF NOT SAFEFLAG2 THEN

    BEGIN
    DOFANGLE[3]:=DOFANGLE[3]-STEPSIZE*PI/180;
    IF DOFANGLE[3]<0 THEN
     DOFANGLE[3]:=0
    END;

   ALPHA:=DOFANGLE;
   ANGLE:=ORIENTATION;
   DISTANCE:=GAP;

   CASE HOLDSTATUS OF
   HOLDNEWWORKPIECE,HOLDOLDWORKPIECE:
    CONFIGURATION(CURRENTPATHSTEP,VARIABLE,K,WKPCPOINTER,FALSE,
                  HOLDOLDWORKPIECE);
   NOWORKPIECE:
    CONFIGURATION(CURRENTPATHSTEP,VARIABLE,K,NIL,FALSE,
                  NOWORKPIECE)
   END

   END

  END

 END

END
```

```
   END;

  J:=J+1;
  PREVIOUSPATHSTEP:=CURRENTPATHSTEP;
  CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR
  END;

 PLOTGRAPHS
 END;

IF NOT SAFEENDS THEN
 SAFEPATH:=FALSE
END;

PROCEDURE GRAPHICS(MANIPULATOR:POINTMANIPULATOR;
                   WKPCNUMBER,STEPNUMBER:INTEGER);

CONST
 DELTA=0.0001;
 XORGIN=4.50;
 YORGIN=5.98;

VAR
 OBSTACLESET,WORKPIECESET:NUMBERSET;
 MANIPULATORSET:SET OF POLYHEDRANAMES;
 CURRADDPLAVERTEX,PREVADDPLAVERTEX:POINTPLANEVERTEX;

PROCEDURE WORKONPOLYHEDRA(PROCEDURE WORK(POLYHEDRON:POINTPOLYHEDRON));

VAR
 CURRPOLYHEDRON:POINTPOLYHEDRON;
 CURRWORKPIECE:POINTDEFINEBOX;

BEGIN
CURRPOLYHEDRON:=FIRSTOBSTACLE;

WHILE CURRPOLYHEDRON<>NIL DO

 BEGIN
 WORK(CURRPOLYHEDRON);
 CURRPOLYHEDRON:=CURRPOLYHEDRON@.NEXTPOLYHEDRON
 END;

CURRWORKPIECE:=FIRSTWORKPIECE;

WHILE CURRWORKPIECE<>NIL DO

 BEGIN
 WITH CURRWORKPIECE@,BOXPOLYHEDRON@ DO
  IF POLYHEDRONNUMBER<>WKPCNUMBER THEN
   WORK(CURRWORKPIECE@.BOXPOLYHEDRON);
 CURRWORKPIECE:=CURRWORKPIECE@.NEXTDEFINEBOX
 END;
```

```
WITH MANIPULATOR@ DO

 BEGIN
 WORK(BASE@.BOXPOLYHEDRON);
 WORK(LINK1@.BOXPOLYHEDRON);
 WORK(LINK2@.BOXPOLYHEDRON);
 WORK(LINK3@.BOXPOLYHEDRON);
 WORK(WRIST@.BOXPOLYHEDRON);
 WORK(FINGER1@.BOXPOLYHEDRON);
 WORK(FINGER2@.BOXPOLYHEDRON);
 IF WKPCNUMBER<>0 THEN
  WORK(WORKPIECE@.BOXPOLYHEDRON)
 END

END;

PROCEDURE FINDVISUALPLANES(POLYHEDRON:POINTPOLYHEDRON);

BEGIN
WITH POLYHEDRON@ DO

 BEGIN
 CURRENTPLANE:=FIRSTPLANE;

 WHILE CURRENTPLANE<>NIL DO

  BEGIN
  WITH CURRENTPLANE@ DO
   IF (OUTWARDNORMAL[1]*GRAPHICSBASIS[3,1]+
   OUTWARDNORMAL[2]*GRAPHICSBASIS[3,2]+OUTWARDNORMAL[3]*
   GRAPHICSBASIS[3,3])>DELTA THEN
    VISIBLEPLANESET:=VISIBLEPLANESET+[PLANENUMBER];
  CURRENTPLANE:=CURRENTPLANE@.NEXTPLANE
  END

 END

END;

PROCEDURE DELETERECORDS(POINTER:POINTPLANEVERTEX);

BEGIN
WITH POINTER@ DO

 BEGIN
 CURRADDPLAVERTEX:=ADDPLANEVERTEX;

 WHILE CURRADDPLAVERTEX<>NIL DO

  BEGIN
  PREVADDPLAVERTEX:=CURRADDPLAVERTEX;
  CURRADDPLAVERTEX:=CURRADDPLAVERTEX@.ADDPLANEVERTEX;
  DISPOSE(PREVADDPLAVERTEX)
  END;
```

```
 ADDPLANEVERTEX:=NIL
 END

END;

PROCEDURE MODIFYPLANES(POLYHEDRON1:POINTPOLYHEDRON);

VAR
 CURRENTPLANE1:POINTPLANE;

PROCEDURE FINDPLANEPAIRS(POLYHEDRON2:POINTPOLYHEDRON);

TYPE
 VERTEXPOSITIONS=(OUTSIDE,BORDER,INSIDE);
 PLANESVISIBILITY=(SEEALLOFPLANE1,SEEALLOFPLANE2,SEEALLOFBOTH,
                    PLANESMAYCOLLIDE,PLANESCOLLIDE);

VAR
 EXIT,INTERSECTION:BOOLEAN;
 CURRENTPLANE2:POINTPLANE;

FUNCTION FINDVISIBILITY:PLANESVISIBILITY;

VAR
 I:1..3;
 MAGNITUDE:REAL;
 VERTEX1,VERTEX2,VERTEX3,VERTEX4:POINT;
 VEC1,VEC2,VEC3,NORMAL:VECTOR;
 COLLISION:BOOLEAN;
 PDVERTEXSET1,ZDVERTEXSET1,
 PDVERTEXSET2,ZDVERTEXSET2:NUMBERSET;

PROCEDURE GENERATESETS(VAR PDVERTEXSET,ZDVERTEXSET:NUMBERSET;
                       PLANE:POINTPLANE;REFERENCE:POINT;
                       NORMAL:VECTOR);

VAR
 I:1..3;
 MAGNITUDE,DISPLACEMENT:REAL;
 VEC:VECTOR;

BEGIN
PDVERTEXSET:=[];
ZDVERTEXSET:=[];
CURRPLANEVERTEX:=PLANE@.PLANEVERTEX;

WHILE CURRPLANEVERTEX<>NIL DO

 BEGIN
 WITH CURRPLANEVERTEX@ DO

  BEGIN
  FOR I:=1 TO 3 DO
```

```
   VEC[I]:=VERTEX[I]-REFERENCE[I];
  MAGNITUDE:=SQRT(SQR(VEC[1])+SQR(VEC[2])+SQR(VEC[3]));
  FOR I:=1 TO 3 DO
   VEC[I]:=VEC[I]/MAGNITUDE;
  DISPLACEMENT:=NORMAL[1]*VEC[1]+NORMAL[2]*VEC[2]+
  NORMAL[3]*VEC[3];
  IF DISPLACEMENT>DELTA THEN
   PDVERTEXSET:=PDVERTEXSET+[VERTEXNUMBER];
  IF ABS(DISPLACEMENT)<=DELTA THEN
   ZDVERTEXSET:=ZDVERTEXSET+[VERTEXNUMBER]
  END;

 CURRPLANEVERTEX:=CURRPLANEVERTEX@.NEXTPLANEVERTEX
 END

END;

PROCEDURE GENERATEVERTICES(PLANEA,PLANEB:POINTPLANE;
                           PDVERTEXSETA,PDVERTEXSETB:NUMBERSET;
                           VAR VERTEXI,VERTEXII:POINT);

VAR
 I:1..3;
 MAGNITUDE:REAL;
 VERTEXIII,VERTEXIV,VERTEXV:POINT;
 VEC1,VEC2,VEC3,VEC4,NORMAL:VECTOR;
 EXITA,EXITB:BOOLEAN;
 CURRPLANEVERTEXA,NEXTVERTEXA,
 CURRPLANEVERTEXB,NEXTVERTEXB:POINTPLANEVERTEX;

BEGIN
WITH PLANEA@ DO

 BEGIN
 CURRPLANEVERTEXA:=PLANEVERTEX;
 EXITA:=FALSE;

 WHILE (CURRPLANEVERTEXA<>NIL) AND NOT EXITA DO

  BEGIN
  WITH CURRPLANEVERTEXA@ DO

   BEGIN
   IF NEXTPLANEVERTEX<>NIL THEN
    NEXTVERTEXA:=NEXTPLANEVERTEX
   ELSE
    NEXTVERTEXA:=PLANEVERTEX;

   IF ((VERTEXNUMBER IN PDVERTEXSETA) AND NOT
   (NEXTVERTEXA@.VERTEXNUMBER IN PDVERTEXSETA))
   OR (NOT (VERTEXNUMBER IN PDVERTEXSETA) AND
   (NEXTVERTEXA@.VERTEXNUMBER IN PDVERTEXSETA))
   THEN
```

```
BEGIN
VERTEXI:=VERTEX;
VERTEXII:=NEXTVERTEXA@.VERTEX;
IF NEXTVERTEXA@.NEXTPLANEVERTEX<>NIL THEN
 VERTEXIII:=NEXTVERTEXA@.NEXTPLANEVERTEX@.VERTEX
ELSE
 VERTEXIII:=PLANEVERTEX@.VERTEX;

FOR I:=1 TO 3 DO

 BEGIN
 VEC1[I]:=VERTEXII[I]-VERTEXI[I];
 VEC2[I]:=VERTEXIII[I]-VERTEXI[I]
 END;

WITH PLANEB@ DO

 BEGIN
 CURRPLANEVERTEXB:=PLANEVERTEX;
 EXITB:=FALSE;

 WHILE (CURRPLANEVERTEXB<>NIL) AND NOT EXITB DO

  BEGIN
  WITH CURRPLANEVERTEXB@ DO

   BEGIN
   IF NEXTPLANEVERTEX<>NIL THEN
    NEXTVERTEXB:=NEXTPLANEVERTEX
   ELSE
    NEXTVERTEXB:=PLANEVERTEX;

   IF ((VERTEXNUMBER IN PDVERTEXSETB) AND NOT
   (NEXTVERTEXB@.VERTEXNUMBER IN PDVERTEXSETB))
   OR (NOT (VERTEXNUMBER IN PDVERTEXSETB) AND
   (NEXTVERTEXB@.VERTEXNUMBER IN PDVERTEXSETB))
   THEN

    BEGIN
    VERTEXIV:=VERTEX;
    VERTEXV:=NEXTVERTEXB@.VERTEX;

    FOR I:=1 TO 3 DO

     BEGIN
     VEC3[I]:=VERTEXIV[I]-VERTEXI[I];
     VEC4[I]:=VERTEXV[I]-VERTEXI[I]
     END;

    NORMAL[1]:=VEC1[2]*VEC3[3]-VEC3[2]*VEC1[3];
    NORMAL[2]:=VEC3[1]*VEC1[3]-VEC1[1]*VEC3[3];
    NORMAL[3]:=VEC1[1]*VEC3[2]-VEC3[1]*VEC1[2];
    MAGNITUDE:=SQRT(SQR(NORMAL[1])+SQR(NORMAL[2])+
    SQR(NORMAL[3]));
```

```
        FOR I:=1 TO 3 DO
         NORMAL[I]:=NORMAL[I]/MAGNITUDE;
        IF (NORMAL[1]*VEC2[1]+NORMAL[2]*VEC2[2]+
        NORMAL[3]*VEC2[3])>0 THEN
         FOR I:=1 TO 3 DO
          NORMAL[I]:=-NORMAL[I];
        IF (NORMAL[1]*VEC4[1]+NORMAL[2]*VEC4[2]+
        NORMAL[3]*VEC4[3])<-DELTA THEN
         EXITB:=TRUE
        END

       END;

      CURRPLANEVERTEXB:=CURRPLANEVERTEXB@.NEXTPLANEVERTEX;

      IF (CURRPLANEVERTEXB=NIL) AND NOT EXITB THEN
       EXITA:=TRUE
      END

     END

    END

   END;

  CURRPLANEVERTEXA:=CURRPLANEVERTEXA@.NEXTPLANEVERTEX
  END

 END;

IF NOT EXITA THEN
 COLLISION:=TRUE
ELSE
 COLLISION:=FALSE
END;

BEGIN
WITH CURRENTPLANE2@,PLANEVERTEX@ DO
 GENERATESETS(PDVERTEXSET1,ZDVERTEXSET1,CURRENTPLANE1,
              VERTEX,OUTWARDNORMAL);
WITH CURRENTPLANE1@,PLANEVERTEX@ DO
 GENERATESETS(PDVERTEXSET2,ZDVERTEXSET2,CURRENTPLANE2,
              VERTEX,OUTWARDNORMAL);
IF ((PDVERTEXSET1+ZDVERTEXSET1=CURRENTPLANE1@.PLANEVERTEXSET)
AND (PDVERTEXSET2+ZDVERTEXSET2=CURRENTPLANE2@.PLANEVERTEXSET))
OR ((PDVERTEXSET1-ZDVERTEXSET1=[])
AND (PDVERTEXSET2-ZDVERTEXSET2=[])) THEN
 FINDVISIBILITY:=SEEALLOFBOTH
ELSE
IF (PDVERTEXSET1+ZDVERTEXSET1=CURRENTPLANE1@.PLANEVERTEXSET)
OR (PDVERTEXSET2-ZDVERTEXSET2=[]) THEN
 FINDVISIBILITY:=SEEALLOFPLANE1
ELSE
IF (PDVERTEXSET2+ZDVERTEXSET2=CURRENTPLANE2@.PLANEVERTEXSET)
```

```
OR (PDVERTEXSET1-ZDVERTEXSET1=[]) THEN
 FINDVISIBILITY:=SEEALLOFPLANE2
ELSE
IF (ZDVERTEXSET1=CURRENTPLANE1@.PLANEVERTEXSET)
OR (ZDVERTEXSET2=CURRENTPLANE2@.PLANEVERTEXSET) THEN
 FINDVISIBILITY:=PLANESMAYCOLLIDE
ELSE

 BEGIN
 GENERATEVERTICES(CURRENTPLANE1,CURRENTPLANE2,PDVERTEXSET1,
                  PDVERTEXSET2,VERTEX1,VERTEX2);

 IF COLLISION THEN
  FINDVISIBILITY:=PLANESCOLLIDE
 ELSE

  BEGIN
  GENERATEVERTICES(CURRENTPLANE2,CURRENTPLANE1,PDVERTEXSET2,
                   PDVERTEXSET1,VERTEX3,VERTEX4);

  FOR I:=1 TO 3 DO

   BEGIN
   VEC1[I]:=VERTEX2[I]-VERTEX1[I];
   VEC2[I]:=VERTEX4[I]-VERTEX3[I];
   VEC3[I]:=VERTEX1[I]-VERTEX3[I]
   END;

  NORMAL[1]:=VEC1[2]*VEC2[3]-VEC2[2]*VEC1[3];
  NORMAL[2]:=VEC2[1]*VEC1[3]-VEC1[1]*VEC2[3];
  NORMAL[3]:=VEC1[1]*VEC2[2]-VEC2[1]*VEC1[2];
  MAGNITUDE:=SQRT(SQR(NORMAL[1])+SQR(NORMAL[2])+SQR(NORMAL[3]));
  FOR I:=1 TO 3 DO
   NORMAL[I]:=NORMAL[I]/MAGNITUDE;
  IF (NORMAL[1]*VEC3[1]+NORMAL[2]*VEC3[2]+NORMAL[3]*VEC3[3])<0 THEN
   FOR I:=1 TO 3 DO
    NORMAL[I]:=-NORMAL[I];
  IF (NORMAL[1]*GRAPHICSBASIS[3,1]+NORMAL[2]*GRAPHICSBASIS[3,2]+
  NORMAL[3]*GRAPHICSBASIS[3,3])>0 THEN
   FINDVISIBILITY:=SEEALLOFPLANE1
  ELSE
   FINDVISIBILITY:=SEEALLOFPLANE2
  END

 END

END;

FUNCTION VERTEXPOSITION(PLANE:POINTPLANE;VERTEX4:PLANEPOINT)
                        :VERTEXPOSITIONS;

VAR
 I:1..2;
 MAGNITUDE,DISPLACEMENT:REAL;
```

```
 VERTEX1,VERTEX2,VERTEX3:PLANEPOINT;
 VEC1,VEC2,VEC3:PLANEVECTOR;
 EXIT:BOOLEAN;

BEGIN
VERTEXPOSITION:=INSIDE;
EXIT:=FALSE;
WITH PLANE@ DO

 BEGIN
 CURRPLANEVERTEX:=PLANEVERTEX;

 WHILE (CURRPLANEVERTEX<>NIL) AND NOT EXIT DO

  BEGIN
  WITH CURRPLANEVERTEX@ DO

   BEGIN
   VERTEX1:=GRAPHICSVERTEX;

   IF NEXTPLANEVERTEX<>NIL THEN
    WITH NEXTPLANEVERTEX@ DO

     BEGIN
     VERTEX2:=GRAPHICSVERTEX;
     IF NEXTPLANEVERTEX<>NIL THEN
      WITH NEXTPLANEVERTEX@ DO
       VERTEX3:=GRAPHICSVERTEX
     ELSE
      VERTEX3:=PLANEVERTEX@.GRAPHICSVERTEX
     END

   ELSE
    WITH PLANEVERTEX@ DO

     BEGIN
     VERTEX2:=GRAPHICSVERTEX;
     VERTEX3:=NEXTPLANEVERTEX@.GRAPHICSVERTEX
     END

   END;

  VEC1[1]:=VERTEX1[2]-VERTEX2[2];
  VEC1[2]:=VERTEX2[1]-VERTEX1[1];
  MAGNITUDE:=SQRT(SQR(VEC1[1])+SQR(VEC1[2]));

  FOR I:=1 TO 2 DO

   BEGIN
   VEC1[I]:=VEC1[I]/MAGNITUDE;
   VEC2[I]:=VERTEX3[I]-VERTEX1[I];
   VEC3[I]:=VERTEX4[I]-VERTEX1[I]
   END;
```

```
  IF (VEC1[1]*VEC2[1]+VEC1[2]*VEC2[2])>0 THEN
   FOR I:=1 TO 2 DO
    VEC1[I]:=-VEC1[I];
  DISPLACEMENT:=VEC1[1]*VEC3[1]+VEC1[2]*VEC3[2];

  IF DISPLACEMENT>DELTA THEN

   BEGIN
   VERTEXPOSITION:=OUTSIDE;
   EXIT:=TRUE
   END

  ELSE
   IF ABS(DISPLACEMENT)<=DELTA THEN
    VERTEXPOSITION:=BORDER;

  CURRPLANEVERTEX:=CURRPLANEVERTEX@.NEXTPLANEVERTEX
  END

 END

END;

PROCEDURE FINDINTERSECTION(FRONTPLANE,BACKPLANE:POINTPLANE);

TYPE
 SLOPETYPES=(HORIZONTAL,REGULAR,VERTICAL);

VAR
 M1,B1,M2,B2:REAL;
 VERTEX1,VERTEX2,VERTEX3,VERTEX4,
 COMMONPOINT,SAVECOMMONPOINT:PLANEPOINT;
 FIRST1,EXIT1,FIRST2,EXIT2,POINT2:BOOLEAN;
 SLOPE1TO2,SLOPE3TO4:SLOPETYPES;
 VERTEX1POSITION,VERTEX2POSITION:VERTEXPOSITIONS;
 BACKVERTEX1,BACKVERTEX2,
 FRONTVERTEX3,FRONTVERTEX4:POINTPLANEVERTEX;

FUNCTION INRANGE(A,B,C:REAL):BOOLEAN;

BEGIN
INRANGE:=FALSE;

IF A>B THEN

 BEGIN
 IF ((A+DELTA)>=C) AND (C>=(B-DELTA)) THEN
  INRANGE:=TRUE
 END

ELSE
 IF ((B+DELTA)>=C) AND (C>=(A-DELTA)) THEN
  INRANGE:=TRUE
END;
```

```
FUNCTION ONEDGE(A,B,C:PLANEPOINT):BOOLEAN;

BEGIN
IF INRANGE(A[1],B[1],C[1]) AND INRANGE(A[2],B[2],C[2]) THEN
 ONEDGE:=TRUE
ELSE
 ONEDGE:=FALSE
END;

PROCEDURE MODIFYEDGE;

VAR
 FIRSTPOINT,SECONDPOINT,
 SAVEPOINT,INSERTPOINT:POINTPLANEVERTEX;

FUNCTION NEWRECORD(COMMONPOINT:PLANEPOINT):POINTPLANEVERTEX;

VAR
 POINTER:POINTPLANEVERTEX;

BEGIN
NEW(POINTER);

WITH POINTER@ DO

 BEGIN
 GRAPHICSVERTEX:=COMMONPOINT;
 DISTANCE:=SQRT(SQR(GRAPHICSVERTEX[1]-VERTEX1[1])+
 SQR(GRAPHICSVERTEX[2]-VERTEX1[2]))
 END;

NEWRECORD:=POINTER
END;

PROCEDURE INSERTRECORD(POINTER:POINTPLANEVERTEX);

VAR
 INSERTNOW:BOOLEAN;

BEGIN
WITH BACKVERTEX1@ DO

 BEGIN
 IF ADDPLANEVERTEX@.DISTANCE>POINTER@.DISTANCE THEN

  BEGIN
  POINTER@.PENDOWN:=PENDOWN;
  POINTER@.ADDPLANEVERTEX:=ADDPLANEVERTEX;
  POINTER@.NEXTPLANEVERTEX:=NIL;
  ADDPLANEVERTEX:=POINTER
  END

 ELSE
```

```
   BEGIN
   PREVADDPLAVERTEX:=ADDPLANEVERTEX;
   CURRADDPLAVERTEX:=PREVADDPLAVERTEX@.ADDPLANEVERTEX;

   WHILE PREVADDPLAVERTEX<>NIL DO

    BEGIN
    IF CURRADDPLAVERTEX<>NIL THEN

     BEGIN
     IF CURRADDPLAVERTEX@.DISTANCE>POINTER@.DISTANCE THEN
      INSERTNOW:=TRUE
     ELSE
      INSERTNOW:=FALSE
     END

    ELSE
     INSERTNOW:=TRUE;

    IF INSERTNOW THEN

     BEGIN
     WITH PREVADDPLAVERTEX@ DO

      BEGIN
      POINTER@.PENDOWN:=PENDOWN;
      POINTER@.ADDPLANEVERTEX:=ADDPLANEVERTEX;
      POINTER@.NEXTPLANEVERTEX:=NEXTPLANEVERTEX;
      ADDPLANEVERTEX:=POINTER;
      NEXTPLANEVERTEX:=NIL
      END;

     PREVADDPLAVERTEX:=NIL
     END

    ELSE

     BEGIN
     PREVADDPLAVERTEX:=CURRADDPLAVERTEX;
     CURRADDPLAVERTEX:=PREVADDPLAVERTEX@.ADDPLANEVERTEX
     END

    END

   END

  END

 END;

 PROCEDURE MODIFYRECORDS1;

 BEGIN
```

```
WITH BACKVERTEX1@ DO
 CASE EDGESTATUS OF

 VISIBLE:

  BEGIN
  EDGESTATUS:=PARTVISIBLE;
  IF VERTEX2POSITION=INSIDE THEN
   PENSTATUS:=RAISEPEN;
  INSERTPOINT:=NEWRECORD(COMMONPOINT);

  WITH INSERTPOINT@ DO

   BEGIN
   PENDOWN:=FALSE;
   ADDPLANEVERTEX:=NIL;
   NEXTPLANEVERTEX:=BACKVERTEX1@.NEXTPLANEVERTEX
   END;

  ADDPLANEVERTEX:=INSERTPOINT
  END;

 PARTVISIBLE:

  BEGIN
  IF VERTEX2POSITION=INSIDE THEN
   PENSTATUS:=RAISEPEN;
  INSERTPOINT:=NEWRECORD(COMMONPOINT);
  INSERTRECORD(INSERTPOINT);
  CURRADDPLAVERTEX:=INSERTPOINT@.ADDPLANEVERTEX;

  WHILE CURRADDPLAVERTEX<>NIL DO

   BEGIN
   PREVADDPLAVERTEX:=CURRADDPLAVERTEX;
   CURRADDPLAVERTEX:=PREVADDPLAVERTEX@.ADDPLANEVERTEX;
   DISPOSE(PREVADDPLAVERTEX)
   END;

  WITH INSERTPOINT@ DO

   BEGIN
   PENDOWN:=FALSE;
   ADDPLANEVERTEX:=NIL;
   NEXTPLANEVERTEX:=BACKVERTEX1@.NEXTPLANEVERTEX
   END

  END;

 INVISIBLE:
  IF VERTEX2POSITION=INSIDE THEN
   PENSTATUS:=RAISEPEN

END;
```

```
EXIT2:=TRUE
END;

PROCEDURE MODIFYRECORDS2;

BEGIN
WITH BACKVERTEX1@ DO
 CASE EDGESTATUS OF

 VISIBLE:

  BEGIN
  EDGESTATUS:=PARTVISIBLE;
  PENSTATUS:=LOWERPEN;
  PENDOWN:=FALSE;
  INSERTPOINT:=NEWRECORD(COMMONPOINT);

  WITH INSERTPOINT@ DO

   BEGIN
   PENDOWN:=TRUE;
   ADDPLANEVERTEX:=NIL;
   NEXTPLANEVERTEX:=BACKVERTEX1@.NEXTPLANEVERTEX
   END;

  ADDPLANEVERTEX:=INSERTPOINT
  END;

 PARTVISIBLE:

  BEGIN
  PENSTATUS:=LOWERPEN;
  INSERTPOINT:=NEWRECORD(COMMONPOINT);
  INSERTRECORD(INSERTPOINT);
  PENDOWN:=FALSE;
  CURRADDPLAVERTEX:=ADDPLANEVERTEX;

  WHILE CURRADDPLAVERTEX<>INSERTPOINT DO

   BEGIN
   PREVADDPLAVERTEX:=CURRADDPLAVERTEX;
   CURRADDPLAVERTEX:=PREVADDPLAVERTEX@.ADDPLANEVERTEX;
   DISPOSE(PREVADDPLAVERTEX)
   END;

  ADDPLANEVERTEX:=INSERTPOINT
  END;

 INVISIBLE:PENSTATUS:=LOWERPEN
 END;

EXIT2:=TRUE
END;
```

```
BEGIN
INTERSECTION:=TRUE;

IF NOT POINT2 THEN

 BEGIN
 VERTEX1POSITION:=VERTEXPOSITION(FRONTPLANE,VERTEX1);
 VERTEX2POSITION:=VERTEXPOSITION(FRONTPLANE,VERTEX2)
 END;

WITH BACKVERTEX1@ DO
 CASE VERTEX1POSITION OF

 OUTSIDE:
  CASE VERTEX2POSITION OF

  OUTSIDE:

   BEGIN
   CASE EDGESTATUS OF

   VISIBLE:
    IF NOT POINT2 THEN
     SAVECOMMONPOINT:=COMMONPOINT
    ELSE

     BEGIN
     EDGESTATUS:=PARTVISIBLE;
     FIRSTPOINT:=NEWRECORD(SAVECOMMONPOINT);
     SECONDPOINT:=NEWRECORD(COMMONPOINT);

     IF FIRSTPOINT@.DISTANCE>SECONDPOINT@.DISTANCE THEN

      BEGIN
      SAVEPOINT:=FIRSTPOINT;
      FIRSTPOINT:=SECONDPOINT;
      SECONDPOINT:=SAVEPOINT
      END;

     WITH FIRSTPOINT@ DO

      BEGIN
      PENDOWN:=FALSE;
      ADDPLANEVERTEX:=SECONDPOINT;
      NEXTPLANEVERTEX:=NIL
      END;

     WITH SECONDPOINT@ DO

      BEGIN
      PENDOWN:=TRUE;
      ADDPLANEVERTEX:=NIL;
      NEXTPLANEVERTEX:=BACKVERTEX1@.NEXTPLANEVERTEX
```

```
     END;

    ADDPLANEVERTEX:=FIRSTPOINT
    END;

  PARTVISIBLE:
   IF NOT POINT2 THEN
    SAVECOMMONPOINT:=COMMONPOINT
   ELSE

    BEGIN
    FIRSTPOINT:=NEWRECORD(SAVECOMMONPOINT);
    SECONDPOINT:=NEWRECORD(COMMONPOINT);

    IF FIRSTPOINT@.DISTANCE>SECONDPOINT@.DISTANCE THEN

     BEGIN
     SAVEPOINT:=FIRSTPOINT;
     FIRSTPOINT:=SECONDPOINT;
     SECONDPOINT:=SAVEPOINT
     END;

    INSERTRECORD(FIRSTPOINT);
    INSERTRECORD(SECONDPOINT);
    FIRSTPOINT@.PENDOWN:=FALSE;
    CURRADDPLAVERTEX:=FIRSTPOINT@.ADDPLANEVERTEX;

    WHILE CURRADDPLAVERTEX<>SECONDPOINT DO

     BEGIN
     PREVADDPLAVERTEX:=CURRADDPLAVERTEX;
     CURRADDPLAVERTEX:=PREVADDPLAVERTEX@.ADDPLANEVERTEX;
     DISPOSE(PREVADDPLAVERTEX)
     END;

    FIRSTPOINT@.ADDPLANEVERTEX:=SECONDPOINT
    END;

  INVISIBLE:
  END;

  IF POINT2 THEN
   EXIT2:=TRUE
  ELSE
   POINT2:=TRUE
  END;

 BORDER,INSIDE:MODIFYRECORDS1
 END;

BORDER:
 CASE VERTEX2POSITION OF

 OUTSIDE:MODIFYRECORDS2;
```

```
  BORDER:

   BEGIN
   EDGESTATUS:=INVISIBLE;
   PENDOWN:=FALSE;
   DELETERECORDS(BACKVERTEX1)
   END;

  INSIDE:MODIFYRECORDS1
  END;

 INSIDE:
  CASE VERTEX2POSITION OF
  OUTSIDE,BORDER:MODIFYRECORDS2;
  INSIDE:
  END

 END

END;

BEGIN
FIRST1:=TRUE;
EXIT1:=FALSE;
BACKVERTEX1:=BACKPLANE@.PLANEVERTEX;

WHILE NOT EXIT1 DO

 BEGIN
 BACKVERTEX2:=BACKVERTEX1@.NEXTPLANEVERTEX;

 IF BACKVERTEX2=NIL THEN

  BEGIN
  EXIT1:=TRUE;
  BACKVERTEX2:=BACKPLANE@.PLANEVERTEX
  END;

 IF FIRST1 THEN

  BEGIN
  FIRST1:=FALSE;
  VERTEX1:=BACKVERTEX1@.GRAPHICSVERTEX;
  VERTEX2:=BACKVERTEX2@.GRAPHICSVERTEX
  END

 ELSE

  BEGIN
  VERTEX1:=VERTEX2;
  VERTEX2:=BACKVERTEX2@.GRAPHICSVERTEX
  END;
```

```
IF ABS(VERTEX1[1]-VERTEX2[1])<=DELTA THEN
 SLOPE1TO2:=VERTICAL
ELSE
 IF ABS(VERTEX1[2]-VERTEX2[2])<=DELTA THEN
  SLOPE1TO2:=HORIZONTAL
 ELSE

  BEGIN
  SLOPE1TO2:=REGULAR;
  M1:=(VERTEX1[2]-VERTEX2[2])/(VERTEX1[1]-VERTEX2[1]);
  B1:=VERTEX1[2]-M1*VERTEX1[1]
  END;

FIRST2:=TRUE;
EXIT2:=FALSE;
POINT2:=FALSE;
FRONTVERTEX3:=FRONTPLANE@.PLANEVERTEX;

WHILE NOT EXIT2 DO

 BEGIN
 FRONTVERTEX4:=FRONTVERTEX3@.NEXTPLANEVERTEX;

 IF FRONTVERTEX4=NIL THEN

  BEGIN
  EXIT2:=TRUE;
  FRONTVERTEX4:=FRONTPLANE@.PLANEVERTEX
  END;

 IF FIRST2 THEN

  BEGIN
  FIRST2:=FALSE;
  VERTEX3:=FRONTVERTEX3@.GRAPHICSVERTEX;
  VERTEX4:=FRONTVERTEX4@.GRAPHICSVERTEX
  END

 ELSE

  BEGIN
  VERTEX3:=VERTEX4;
  VERTEX4:=FRONTVERTEX4@.GRAPHICSVERTEX
  END;

 IF ABS(VERTEX3[1]-VERTEX4[1])<=DELTA THEN
  SLOPE3TO4:=VERTICAL
 ELSE
  IF ABS(VERTEX3[2]-VERTEX4[2])<=DELTA THEN
   SLOPE3TO4:=HORIZONTAL
  ELSE

   BEGIN
   SLOPE3TO4:=REGULAR;
```

```
  M2:=(VERTEX3[2]-VERTEX4[2])/(VERTEX3[1]-VERTEX4[1]);
  B2:=VERTEX3[2]-M2*VERTEX3[1]
  END;

CASE SLOPE1TO2 OF

HORIZONTAL:
 CASE SLOPE3TO4 OF
 HORIZONTAL:;

 REGULAR:

  BEGIN
  COMMONPOINT[2]:=VERTEX1[2];
  COMMONPOINT[1]:=(COMMONPOINT[2]-B2)/M2;
  IF INRANGE(VERTEX1[1],VERTEX2[1],COMMONPOINT[1])
  AND ONEDGE(VERTEX3,VERTEX4,COMMONPOINT) THEN
   MODIFYEDGE
  END;

 VERTICAL:

  BEGIN
  COMMONPOINT[1]:=VERTEX3[1];
  COMMONPOINT[2]:=VERTEX1[2];
  IF INRANGE(VERTEX1[1],VERTEX2[1],COMMONPOINT[1])
  AND INRANGE(VERTEX3[2],VERTEX4[2],COMMONPOINT[2]) THEN
   MODIFYEDGE
  END

 END;

REGULAR:
 CASE SLOPE3TO4 OF

  HORIZONTAL:

   BEGIN
   COMMONPOINT[2]:=VERTEX3[2];
   COMMONPOINT[1]:=(COMMONPOINT[2]-B1)/M1;
   IF ONEDGE(VERTEX1,VERTEX2,COMMONPOINT)
   AND INRANGE(VERTEX3[1],VERTEX4[1],COMMONPOINT[1]) THEN
    MODIFYEDGE
   END;

  REGULAR:
   IF ABS(M1-M2)>DELTA THEN

    BEGIN
    COMMONPOINT[1]:=(B2-B1)/(M1-M2);
    COMMONPOINT[2]:=M1*COMMONPOINT[1]+B1;
    IF ONEDGE(VERTEX1,VERTEX2,COMMONPOINT)
    AND ONEDGE(VERTEX3,VERTEX4,COMMONPOINT) THEN
     MODIFYEDGE
```

```
      END;

     VERTICAL:

      BEGIN
      COMMONPOINT[1]:=VERTEX3[1];
      COMMONPOINT[2]:=M1*COMMONPOINT[1]+B1;
      IF ONEDGE(VERTEX1,VERTEX2,COMMONPOINT)
      AND INRANGE(VERTEX3[2],VERTEX4[2],COMMONPOINT[2]) THEN
       MODIFYEDGE
      END

     END;

    VERTICAL:
     CASE SLOPE3TO4 OF

     HORIZONTAL:

      BEGIN
      COMMONPOINT[1]:=VERTEX1[1];
      COMMONPOINT[2]:=VERTEX3[2];
      IF INRANGE(VERTEX1[2],VERTEX2[2],COMMONPOINT[2])
      AND INRANGE(VERTEX3[1],VERTEX4[1],COMMONPOINT[1]) THEN
       MODIFYEDGE
      END;

     REGULAR:

      BEGIN
      COMMONPOINT[1]:=VERTEX1[1];
      COMMONPOINT[2]:=M2*COMMONPOINT[1]+B2;
      IF INRANGE(VERTEX1[2],VERTEX2[2],COMMONPOINT[2])
      AND ONEDGE(VERTEX3,VERTEX4,COMMONPOINT) THEN
       MODIFYEDGE
      END;

     VERTICAL:
     END

    END;

   FRONTVERTEX3:=FRONTVERTEX4
   END;

  BACKVERTEX1:=BACKVERTEX2
  END

 END;

 PROCEDURE DELETELINES(FRONTPLANE,BACKPLANE:POINTPLANE);

 VAR
  OVERLAPREGION:BOOLEAN;
```

```
 CURRBACKVERTEX:POINTPLANEVERTEX;

BEGIN
OVERLAPREGION:=FALSE;
CURRBACKVERTEX:=BACKPLANE@.PLANEVERTEX;

WHILE CURRBACKVERTEX<>NIL DO

 BEGIN
 WITH CURRBACKVERTEX@ DO
  CASE EDGESTATUS OF

  VISIBLE:
   IF OVERLAPREGION THEN

    BEGIN
    EDGESTATUS:=INVISIBLE;
    PENDOWN:=FALSE
    END;

  PARTVISIBLE:
   CASE PENSTATUS OF
   RAISEPEN:OVERLAPREGION:=TRUE;
   LOWERPEN:OVERLAPREGION:=FALSE;
   HOLDPEN:
    IF OVERLAPREGION THEN

     BEGIN
     EDGESTATUS:=INVISIBLE;
     PENDOWN:=FALSE;
     DELETERECORDS(CURRBACKVERTEX)
     END

   END;

  INVISIBLE:
   CASE PENSTATUS OF
   RAISEPEN:OVERLAPREGION:=TRUE;
   LOWERPEN:OVERLAPREGION:=FALSE;
   HOLDPEN:
   END

  END;

 CURRBACKVERTEX:=CURRBACKVERTEX@.NEXTPLANEVERTEX
 END;

CURRBACKVERTEX:=BACKPLANE@.PLANEVERTEX;

WHILE CURRBACKVERTEX<>NIL DO

 BEGIN
 WITH CURRBACKVERTEX@ DO
  CASE EDGESTATUS OF
```

```
    VISIBLE:
     IF OVERLAPREGION THEN

      BEGIN
      EDGESTATUS:=INVISIBLE;
      PENDOWN:=FALSE
      END;

    PARTVISIBLE:
     CASE PENSTATUS OF
     RAISEPEN:PENSTATUS:=HOLDPEN;
     LOWERPEN:

      BEGIN
      OVERLAPREGION:=FALSE;
      PENSTATUS:=HOLDPEN
      END;

     HOLDPEN:
      IF OVERLAPREGION THEN

       BEGIN
       EDGESTATUS:=INVISIBLE;
       PENDOWN:=FALSE;
       DELETERECORDS(CURRBACKVERTEX)
       END

     END;

    INVISIBLE:
     CASE PENSTATUS OF
     RAISEPEN:PENSTATUS:=HOLDPEN;
     LOWERPEN:

      BEGIN
      OVERLAPREGION:=FALSE;
      PENSTATUS:=HOLDPEN
      END;

     HOLDPEN:
     END

    END;

  CURRBACKVERTEX:=CURRBACKVERTEX@.NEXTPLANEVERTEX
  END

END;

PROCEDURE PLANEINSIDE(FRONTPLANE,BACKPLANE:POINTPLANE;CODE:STRING(3));

VAR
 VERTEX4:PLANEPOINT;
```

```
 EXIT,OMITBACKPLANE:BOOLEAN;
 CURRBACKVERTEX:POINTPLANEVERTEX;

BEGIN
OMITBACKPLANE:=TRUE;
EXIT:=FALSE;
CURRBACKVERTEX:=BACKPLANE@.PLANEVERTEX;

WHILE (CURRBACKVERTEX<>NIL) AND NOT EXIT DO

 BEGIN
 VERTEX4:=CURRBACKVERTEX@.GRAPHICSVERTEX;

 IF VERTEXPOSITION(FRONTPLANE,VERTEX4)=OUTSIDE THEN

  BEGIN
  OMITBACKPLANE:=FALSE;
  EXIT:=TRUE
  END;

 CURRBACKVERTEX:=CURRBACKVERTEX@.NEXTPLANEVERTEX
 END;

IF OMITBACKPLANE THEN

 BEGIN
 IF CODE='ONE' THEN
  WITH POLYHEDRON1@,CURRENTPLANE1@ DO

   BEGIN
   VISIBLEPLANESET:=VISIBLEPLANESET-[PLANENUMBER];
   CURRBACKVERTEX:=PLANEVERTEX
   END

 ELSE
  WITH POLYHEDRON2@,CURRENTPLANE2@ DO

   BEGIN
   VISIBLEPLANESET:=VISIBLEPLANESET-[PLANENUMBER];
   CURRBACKVERTEX:=PLANEVERTEX
   END;

 WHILE CURRBACKVERTEX<>NIL DO

  BEGIN
  WITH CURRBACKVERTEX@ DO

   BEGIN
   EDGESTATUS:=VISIBLE;
   PENSTATUS:=HOLDPEN;
   PENDOWN:=TRUE;
   DELETERECORDS(CURRBACKVERTEX)
   END;
```

```
  CURRBACKVERTEX:=CURRBACKVERTEX@.NEXTPLANEVERTEX
  END

 END

END;

BEGIN
EXIT:=FALSE;
WITH POLYHEDRON2@ DO

 BEGIN

 CASE POLYHEDRONTYPE OF
 OBSTACLE:
  IF POLYHEDRONNUMBER IN OBSTACLESET THEN
   EXIT:=TRUE;
 FREEWORKPIECE:
  IF POLYHEDRONNUMBER IN WORKPIECESET THEN
   EXIT:=TRUE;
 BASENAME..HELDWORKPIECE:
  IF POLYHEDRONTYPE IN MANIPULATORSET THEN
   EXIT:=TRUE
 END;

 IF NOT EXIT THEN

  BEGIN
  CURRENTPLANE2:=FIRSTPLANE;

  WHILE CURRENTPLANE2<>NIL DO

   BEGIN
   IF CURRENTPLANE2@.PLANENUMBER IN VISIBLEPLANESET THEN

    BEGIN
    INTERSECTION:=FALSE;

    CASE FINDVISIBILITY OF

    SEEALLOFPLANE1:

     BEGIN
     FINDINTERSECTION(CURRENTPLANE1,CURRENTPLANE2);
     IF INTERSECTION THEN
      DELETELINES(CURRENTPLANE1,CURRENTPLANE2)
     ELSE
      PLANEINSIDE(CURRENTPLANE1,CURRENTPLANE2,'TWO')
     END;

    SEEALLOFPLANE2:

     BEGIN
     FINDINTERSECTION(CURRENTPLANE2,CURRENTPLANE1);
```

```
      IF INTERSECTION THEN
       DELETELINES(CURRENTPLANE2,CURRENTPLANE1)
      ELSE
       PLANEINSIDE(CURRENTPLANE2,CURRENTPLANE1,'ONE')
      END;

     SEEALLOFBOTH,PLANESMAYCOLLIDE,PLANESCOLLIDE:
     END

     END;

    CURRENTPLANE2:=CURRENTPLANE2@.NEXTPLANE
    END

   END

  END

 END;

 BEGIN
 WITH POLYHEDRON1@ DO

  BEGIN

  CASE POLYHEDRONTYPE OF
  OBSTACLE:OBSTACLESET:=OBSTACLESET+[POLYHEDRONNUMBER];
  FREEWORKPIECE:WORKPIECESET:=WORKPIECESET+[POLYHEDRONNUMBER];
  BASENAME..HELDWORKPIECE:
   MANIPULATORSET:=MANIPULATORSET+[POLYHEDRONTYPE]
  END;

  CURRENTPLANE1:=FIRSTPLANE;

  WHILE CURRENTPLANE1<>NIL DO

   BEGIN
   WITH CURRENTPLANE1@ DO
    IF PLANENUMBER IN VISIBLEPLANESET THEN
     WORKONPOLYHEDRA(FINDPLANEPAIRS);
   CURRENTPLANE1:=CURRENTPLANE1@.NEXTPLANE
   END

  END

 END;

 PROCEDURE PLOTPOLYHEDRON(POLYHEDRON:POINTPOLYHEDRON);

 PROCEDURE PLOTPLANE;

 VAR
  J:2..3;
```

```
FUNCTION SCALE(COORDINATE:REAL):REAL;

BEGIN
IF ABS(COORDINATE)<=DELTA THEN
 SCALE:=0
ELSE
 SCALE:=0.03*COORDINATE
END;

BEGIN
WITH CURRENTPLANE@ DO

 BEGIN
 CURRADDPLAVERTEX:=NIL;
 CURRPLANEVERTEX:=PLANEVERTEX;
 J:=3;

 WHILE (CURRADDPLAVERTEX<>NIL) OR (CURRPLANEVERTEX<>NIL) DO
  IF CURRADDPLAVERTEX<>NIL THEN

   BEGIN
   WITH CURRADDPLAVERTEX@ DO

    BEGIN
    IF J=2 THEN
     PLOT(SCALE(GRAPHICSVERTEX[1]),SCALE(GRAPHICSVERTEX[2]),2)
    ELSE
     IF PENDOWN THEN
      PLOT(SCALE(GRAPHICSVERTEX[1]),SCALE(GRAPHICSVERTEX[2]),3);
    IF PENDOWN THEN
     J:=2
    ELSE
     J:=3
    END;

   CURRPLANEVERTEX:=CURRADDPLAVERTEX@.NEXTPLANEVERTEX;
   CURRADDPLAVERTEX:=CURRADDPLAVERTEX@.ADDPLANEVERTEX
   END

  ELSE

   BEGIN
   WITH CURRPLANEVERTEX@ DO

    BEGIN
    IF J=2 THEN
     PLOT(SCALE(GRAPHICSVERTEX[1]),SCALE(GRAPHICSVERTEX[2]),2)
    ELSE
     IF PENDOWN THEN
      PLOT(SCALE(GRAPHICSVERTEX[1]),SCALE(GRAPHICSVERTEX[2]),3);
    IF PENDOWN THEN
     J:=2
    ELSE
     J:=3
```

```
    END;

   CURRADDPLAVERTEX:=CURRPLANEVERTEX@.ADDPLANEVERTEX;
   CURRPLANEVERTEX:=CURRPLANEVERTEX@.NEXTPLANEVERTEX
   END;

  WITH PLANEVERTEX@ DO
   PLOT(SCALE(GRAPHICSVERTEX[1]),SCALE(GRAPHICSVERTEX[2]),J)
 END

END;

BEGIN
WITH POLYHEDRON@ DO

 BEGIN
 CURRENTPLANE:=FIRSTPLANE;

 WHILE CURRENTPLANE<>NIL DO

  BEGIN
  WITH CURRENTPLANE@ DO
   IF PLANENUMBER IN VISIBLEPLANESET THEN
    PLOTPLANE;
  CURRENTPLANE:=CURRENTPLANE@.NEXTPLANE
  END

 END

END;

PROCEDURE RESTOREPLANES(POLYHEDRON:POINTPOLYHEDRON);

BEGIN
WITH POLYHEDRON@ DO

 BEGIN
 CURRENTPLANE:=FIRSTPLANE;

 WHILE CURRENTPLANE<>NIL DO

  BEGIN
  WITH CURRENTPLANE@ DO
   IF PLANENUMBER IN VISIBLEPLANESET THEN

    BEGIN
    CURRPLANEVERTEX:=PLANEVERTEX;

    WHILE CURRPLANEVERTEX<>NIL DO

     BEGIN
     WITH CURRPLANEVERTEX@ DO

      BEGIN
```

```
      EDGESTATUS:=VISIBLE;
      PENSTATUS:=HOLDPEN;
      PENDOWN:=TRUE;
      DELETERECORDS(CURRPLANEVERTEX)
      END;

     CURRPLANEVERTEX:=CURRPLANEVERTEX@.NEXTPLANEVERTEX
     END

   END;

  CURRENTPLANE:=CURRENTPLANE@.NEXTPLANE
  END;

 VISIBLEPLANESET:=[]
 END

END;

BEGIN
PLOT(4.5,6.65,-3);
PLOT(3,3.355,3);
PLOT(-3,3.355,2);
PLOT(-3,-3.355,2);
PLOT(3,-3.355,2);
PLOT(3,3.355,2);
PLOT(-4.5,-6.65,-3);

WORKONPOLYHEDRA(FINDVISUALPLANES);

OBSTACLESET:=[];
WORKPIECESET:=[];
MANIPULATORSET:=[];
WORKONPOLYHEDRA(MODIFYPLANES);

PLOT(XORGIN,YORGIN,-3);
WORKONPOLYHEDRA(PLOTPOLYHEDRON);
PLOT(-XORGIN,-YORGIN,-3);

WORKONPOLYHEDRA(RESTOREPLANES);

MESSAGE:='PATH';
SYMBOL(3.15,2.4,0.15,MESSAGE,0,4);
PRINTINTEGER(PATHNUMBER,4.05,2.4,0.15);
MESSAGE:='STEP';
SYMBOL(4.8,2.4,0.15,MESSAGE,0,4);
PRINTINTEGER(STEPNUMBER,5.7,2.4,0.15);

WITH MANIPULATOR@ DO
 CASE MANIPULATORTYPE OF

 FIXED:
  IF SAFEMANIPULATOR THEN
```

```
    BEGIN
    MESSAGE:='SAFE FIXED CONFIGURATION';
    SYMBOL(2.7,2.1,0.15,MESSAGE,0,24)
    END

   ELSE

    BEGIN
    MESSAGE:='UNSAFE FIXED CONFIGURATION';
    SYMBOL(2.55,2.1,0.15,MESSAGE,0,26)
    END;

  VARIABLE:
   IF SAFEMANIPULATOR THEN

    BEGIN
    MESSAGE:='SAFE VARIABLE CONFIGURATION';
    SYMBOL(2.475,2.1,0.15,MESSAGE,0,27)
    END

   ELSE

    BEGIN
    MESSAGE:='UNSAFE VARIABLE CONFIGURATION';
    SYMBOL(2.325,2.1,0.15,MESSAGE,0,29)
    END

  END;

 IF SAFEPATH THEN

  BEGIN
  MESSAGE:='PATH IS SAFE';
  SYMBOL(3.6,1.8,0.15,MESSAGE,0,12)
  END

 ELSE

  BEGIN
  MESSAGE:='PATH IS UNSAFE, SEE DA=PROCCA.ERRORS';
  SYMBOL(1.8,1.8,0.15,MESSAGE,0,36)
  END;

 PRINTPAGENUMBER;
 PLOT(0,0,999)
 END;

 PROCEDURE PRODUCEPATH(POINTER1:POINTDEFINEBOX;
                       HOLDSTATUS:HOLDSTATUSNAMES);

 CONST
  DELTA=0.001;

 VAR
```

```
 I:1..3;
 J:INTEGER;
 BASEANGLE1,BASEANGLE2,DIFFERENCE,DELTAK,WRISTADJUST:REAL;
 LOCATION1,LOCATION2:POINT;
 BASENOCHANGE,NORMAL,POSITIVEADJUST,WRISTNOCHANGE:BOOLEAN;
 DOFNOCHANGE:ARRAY[2..3] OF BOOLEAN;
 POSTPATHPOINTER1,PREPATHPOINTER2:POINTMANIPULATOR;

PROCEDURE FINDADJUSTMENT;

BEGIN
DIFFERENCE:=BASEANGLE2-BASEANGLE1;
IF (ABS(DIFFERENCE)<=DELTA) OR (ABS(ABS(DIFFERENCE)-2*PI)<=DELTA) THEN
 BASENOCHANGE:=TRUE
ELSE

 BEGIN
 BASENOCHANGE:=FALSE;
 NORMAL:=FALSE;

 IF (BASEANGLE1>(95*PI/180)) AND ((85*PI/180)>BASEANGLE2) THEN

  BEGIN
  POSITIVEADJUST:=TRUE;
  DELTAK:=(5*PI/180)/(BASEANGLE2+2*PI-BASEANGLE1)
  END

 ELSE
  IF (BASEANGLE2>(95*PI/180)) AND ((85*PI/180)>BASEANGLE1) THEN

   BEGIN
   POSITIVEADJUST:=FALSE;
   DELTAK:=(-5*PI/180)/(BASEANGLE2-2*PI-BASEANGLE1)
   END

  ELSE

   BEGIN
   NORMAL:=TRUE;
   DELTAK:=ABS((5*PI/180)/(BASEANGLE2-BASEANGLE1))
   END

 END

END;

FUNCTION INRANGE(LOWER,UPPER,THETA:REAL;NORMAL:BOOLEAN):BOOLEAN;

BEGIN
IF ((LOWER<THETA) AND (THETA<UPPER))
OR ((LOWER>THETA) AND (THETA>UPPER)) THEN

 BEGIN
 IF NORMAL THEN
```

```
  INRANGE:=TRUE
 ELSE
  INRANGE:=FALSE
 END

ELSE

 BEGIN
 IF NORMAL THEN
  INRANGE:=FALSE
 ELSE
  INRANGE:=TRUE
 END

END;

PROCEDURE CHECKPOSITION(POINTER2:POINTPOLYHEDRON;RANGE2ONLY:BOOLEAN);

VAR
 LOWERBOUND,UPPERBOUND,LOWERALPHA,UPPERALPHA:REAL;
 INSERT:BOOLEAN;

PROCEDURE FINDBOUNDS(LOWERBOUNDX,UPPERBOUNDX:REAL;
                     NORMALBOUNDSX:BOOLEAN);

VAR
 BASEANGLE1FLAG,BASEANGLE2FLAG,
 LOWERBOUNDFLAG,UPPERBOUNDFLAG:BOOLEAN;

BEGIN
BASEANGLE1FLAG:=INRANGE(LOWERBOUNDX,UPPERBOUNDX,BASEANGLE1,
                        NORMALBOUNDSX);
BASEANGLE2FLAG:=INRANGE(LOWERBOUNDX,UPPERBOUNDX,BASEANGLE2,
                        NORMALBOUNDSX);
LOWERBOUNDFLAG:=INRANGE(BASEANGLE1,BASEANGLE2,LOWERBOUNDX,NORMAL);
UPPERBOUNDFLAG:=INRANGE(BASEANGLE1,BASEANGLE2,UPPERBOUNDX,NORMAL);

IF BASEANGLE1FLAG OR BASEANGLE2FLAG
OR LOWERBOUNDFLAG OR UPPERBOUNDFLAG THEN

 BEGIN
 INSERT:=TRUE;

 IF BASEANGLE1FLAG AND BASEANGLE2FLAG THEN

  BEGIN
  IF BASEANGLE1<BASEANGLE2 THEN

   BEGIN
   LOWERBOUND:=BASEANGLE1;
   UPPERBOUND:=BASEANGLE2
   END

  ELSE
```

```
    BEGIN
    LOWERBOUND:=BASEANGLE2;
    UPPERBOUND:=BASEANGLE1
    END

   END

  ELSE

   BEGIN
   IF LOWERBOUNDFLAG THEN
    LOWERBOUND:=LOWERBOUNDX
   ELSE
    IF BASEANGLE1FLAG THEN
     LOWERBOUND:=BASEANGLE1
    ELSE
     LOWERBOUND:=BASEANGLE2;
   IF UPPERBOUNDFLAG THEN
    UPPERBOUND:=UPPERBOUNDX
   ELSE
    IF BASEANGLE1FLAG THEN
     UPPERBOUND:=BASEANGLE1
    ELSE
     UPPERBOUND:=BASEANGLE2
   END

  END

END;

PROCEDURE INSERTRECORDS;

PROCEDURE INSERTRECORD(FIRSTINSERT:BOOLEAN);

VAR
 I:2..3;
 KVALUE:REAL;
 EXIT:BOOLEAN;
 PATHINSERT:POINTMANIPULATOR;

BEGIN
IF NORMAL THEN
 KVALUE:=(ALPHA[1]-BASEANGLE1)/(BASEANGLE2-BASEANGLE1)
ELSE
 IF POSITIVEADJUST THEN

  BEGIN
  IF ALPHA[1]>(95*PI/180) THEN
   KVALUE:=(ALPHA[1]-BASEANGLE1)/(BASEANGLE2+2*PI-BASEANGLE1)
  ELSE
   KVALUE:=(ALPHA[1]+2*PI-BASEANGLE1)/(BASEANGLE2+2*PI-BASEANGLE1)
  END
```

```
 ELSE

  BEGIN
  IF ALPHA[1]<(95*PI/180) THEN
   KVALUE:=(ALPHA[1]-BASEANGLE1)/(BASEANGLE2-2*PI-BASEANGLE1)
  ELSE
   KVALUE:=(ALPHA[1]-2*PI-BASEANGLE1)/(BASEANGLE2-2*PI-BASEANGLE1)
  END;

EXIT:=FALSE;
PREVIOUSPATHSTEP:=PATHPOINTER1;
CURRENTPATHSTEP:=PATHPOINTER1@.NEXTMANIPULATOR;

WHILE NOT EXIT DO

 BEGIN
 IF CURRENTPATHSTEP@.K>KVALUE THEN

  BEGIN
  EXIT:=TRUE;
  IF ((CURRENTPATHSTEP@.K-PREVIOUSPATHSTEP@.K)>DELTAK)
  OR FIRSTINSERT THEN

   BEGIN
   FOR I:=2 TO 3 DO
    IF DOFNOCHANGE[I] THEN
     ALPHA[I]:=PATHPOINTER1@.DOFANGLE[I]
    ELSE
     ALPHA[I]:=KVALUE*(PATHPOINTER2@.DOFANGLE[I]-PATHPOINTER1@.
     DOFANGLE[I])+PATHPOINTER1@.DOFANGLE[I];
   IF WRISTNOCHANGE THEN
    ANGLE:=PATHPOINTER1@.ORIENTATION
   ELSE
    ANGLE:=KVALUE*(PATHPOINTER2@.ORIENTATION+WRISTADJUST-
    PATHPOINTER1@.ORIENTATION)+PATHPOINTER1@.ORIENTATION;
   DISTANCE:=KVALUE*(PATHPOINTER2@.GAP-PATHPOINTER1@.GAP)+
   PATHPOINTER1@.GAP;
   CONFIGURATION(PATHINSERT,VARIABLE,KVALUE,POINTER1,TRUE,HOLDSTATUS);
   PATHINSERT@.NEXTMANIPULATOR:=CURRENTPATHSTEP;
   PREVIOUSPATHSTEP@.NEXTMANIPULATOR:=PATHINSERT
   END

  END;

 PREVIOUSPATHSTEP:=CURRENTPATHSTEP;
 CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR
 END

END;

BEGIN
IF (UPPERBOUND-LOWERBOUND)<PI THEN

 BEGIN
```

```
 ALPHA[1]:=(LOWERBOUND+UPPERBOUND)/2;
 INSERTRECORD(TRUE);
 LOWERALPHA:=ALPHA[1];
 UPPERALPHA:=ALPHA[1];

 WHILE (LOWERALPHA-LOWERBOUND)>(5*PI/180) DO

  BEGIN
  LOWERALPHA:=LOWERALPHA-5*PI/180;
  ALPHA[1]:=LOWERALPHA;
  INSERTRECORD(FALSE);
  UPPERALPHA:=UPPERALPHA+5*PI/180;
  ALPHA[1]:=UPPERALPHA;
  INSERTRECORD(FALSE)
  END

 END

ELSE

 BEGIN
 ALPHA[1]:=ADJUSTANGLE((LOWERBOUND+UPPERBOUND)/2-PI);
 INSERTRECORD(TRUE);
 LOWERALPHA:=ALPHA[1];
 UPPERALPHA:=ALPHA[1];

 IF ALPHA[1]>0 THEN
  WHILE (UPPERALPHA-UPPERBOUND)>(5*PI/180) DO

   BEGIN
   LOWERALPHA:=ADJUSTANGLE(LOWERALPHA+5*PI/180);
   ALPHA[1]:=LOWERALPHA;
   INSERTRECORD(FALSE);
   UPPERALPHA:=UPPERALPHA-5*PI/180;
   ALPHA[1]:=UPPERALPHA;
   INSERTRECORD(FALSE)
   END

 ELSE
  WHILE (LOWERALPHA-LOWERBOUND)<(5*PI/180) DO

   BEGIN
   LOWERALPHA:=LOWERALPHA+5*PI/180;
   ALPHA[1]:=LOWERALPHA;
   INSERTRECORD(FALSE);
   UPPERALPHA:=ADJUSTANGLE(UPPERALPHA-5*PI/180);
   ALPHA[1]:=UPPERALPHA;
   INSERTRECORD(FALSE)
   END

 END

END;
```

```
BEGIN
WITH POINTER2@ DO

 BEGIN
 IF NOT RANGE2ONLY THEN

  BEGIN
  INSERT:=FALSE;
  FINDBOUNDS(LOWERBOUND1,UPPERBOUND1,NORMALBOUNDS1);
  IF INSERT THEN
   INSERTRECORDS
  END;

 INSERT:=FALSE;
 FINDBOUNDS(LOWERBOUND2,UPPERBOUND2,NORMALBOUNDS2);
 IF INSERT THEN
  INSERTRECORDS
 END

END;

BEGIN
BASEANGLE1:=PATHPOINTER1@.DOFANGLE[1];
BASEANGLE2:=PATHPOINTER2@.DOFANGLE[1];
FINDADJUSTMENT;

IF BASENOCHANGE THEN

 BEGIN
 WRITELN(ERRORS,'NO CHANGE IN BASEANGLES *':25);
 GOTO 1
 END;

DIFFERENCE:=PATHPOINTER2@.ORIENTATION-PATHPOINTER1@.ORIENTATION;

IF (ABS(DIFFERENCE)<DELTA) OR (ABS(DIFFERENCE-PI)<DELTA) THEN
 WRISTNOCHANGE:=TRUE
ELSE

 BEGIN
 WRISTNOCHANGE:=FALSE;
 IF (DIFFERENCE>PI/2) THEN

  BEGIN
  IF (DIFFERENCE<3*PI/2) THEN
   WRISTADJUST:=-PI
  ELSE
   WRISTADJUST:=-2*PI
  END

 ELSE
  IF (DIFFERENCE<-PI/2) THEN

   BEGIN
```

```
   IF (DIFFERENCE>-3*PI/2) THEN
    WRISTADJUST:=PI
   ELSE
    WRISTADJUST:=2*PI
   END

  ELSE
   WRISTADJUST:=0
  END;

CASE HOLDSTATUS OF
HOLDNEWWORKPIECE,HOLDOLDWORKPIECE:;

NOWORKPIECE:

 BEGIN
 LOCATION1:=PATHPOINTER1@.WRIST@.REFERENCE;
 LOCATION2:=PATHPOINTER2@.WRIST@.REFERENCE;

 IF (LOCATION1[3]<LOCATION2[3]) AND NOT LASTPATH THEN

  BEGIN
  WITH WKPCPOINTER@,BOXPOLYHEDRON@ DO
   IF INRANGE(BASEANGLE1,BASEANGLE2,UPPERBOUND1,NORMAL) THEN
    ALPHA[1]:=UPPERBOUND1
   ELSE
    ALPHA[1]:=LOWERBOUND1;
  FOR I:=2 TO 3 DO
   ALPHA[I]:=PATHPOINTER2@.DOFANGLE[I];
  ANGLE:=PATHPOINTER2@.ORIENTATION;
  DISTANCE:=PATHPOINTER2@.GAP;
  CONFIGURATION(PREPATHPOINTER2,FIXED,1,NIL,TRUE,NOWORKPIECE);
  PREPATHPOINTER2@.NEXTMANIPULATOR:=PATHPOINTER2;
  PATHPOINTER2:=PREPATHPOINTER2;
  BASEANGLE2:=PATHPOINTER2@.DOFANGLE[1];
  FINDADJUSTMENT
  END;

 IF (LOCATION1[3]>LOCATION2[3]) AND NOT FIRSTPATH THEN

  BEGIN
  WITH PREVWKPCPOINTER@,BOXPOLYHEDRON@ DO
   IF INRANGE(BASEANGLE1,BASEANGLE2,UPPERBOUND1,NORMAL) THEN
    ALPHA[1]:=UPPERBOUND1
   ELSE
    ALPHA[1]:=LOWERBOUND1;
  FOR I:=2 TO 3 DO
   ALPHA[I]:=PATHPOINTER1@.DOFANGLE[I];
  ANGLE:=PATHPOINTER1@.ORIENTATION;
  DISTANCE:=PATHPOINTER1@.GAP;
  CONFIGURATION(POSTPATHPOINTER1,FIXED,0,NIL,TRUE,NOWORKPIECE);
  PATHPOINTER1@.NEXTMANIPULATOR:=POSTPATHPOINTER1;
  PATHPOINTER1:=POSTPATHPOINTER1;
  BASEANGLE1:=PATHPOINTER1@.DOFANGLE[1];
```

```
  FINDADJUSTMENT
  END

 END

END;

FOR I:=2 TO 3 DO
 IF ABS(PATHPOINTER2@.DOFANGLE[I]-PATHPOINTER1@.DOFANGLE[I])<DELTA
 THEN
  DOFNOCHANGE[I]:=TRUE
 ELSE
  DOFNOCHANGE[I]:=FALSE;

PATHPOINTER1@.NEXTMANIPULATOR:=PATHPOINTER2;

IF NOT BASENOCHANGE THEN

 BEGIN
 CURRPOLYHEDRON:=FIRSTOBSTACLE;

 WHILE CURRPOLYHEDRON<>NIL DO

  BEGIN
  CHECKPOSITION(CURRPOLYHEDRON,FALSE);
  CURRPOLYHEDRON:=CURRPOLYHEDRON@.NEXTPOLYHEDRON
  END;

 CURRWORKPIECE:=FIRSTWORKPIECE;

 WHILE CURRWORKPIECE<>NIL DO

  BEGIN
  WITH CURRWORKPIECE@,BOXPOLYHEDRON@ DO
   CASE HOLDSTATUS OF
   HOLDNEWWORKPIECE,HOLDOLDWORKPIECE:
    IF POLYHEDRONNUMBER<>WKPCNUMBER THEN
     CHECKPOSITION(BOXPOLYHEDRON,POLYHEDRONNUMBER=PREVWKPCNUMBER);
   NOWORKPIECE:
    CHECKPOSITION(BOXPOLYHEDRON,(POLYHEDRONNUMBER=PREVWKPCNUMBER)
                  OR (POLYHEDRONNUMBER=WKPCNUMBER))
   END;

  CURRWORKPIECE:=CURRWORKPIECE@.NEXTDEFINEBOX
  END

 END;

CURRENTPATHSTEP:=PATHSTART;

WHILE CURRENTPATHSTEP<>NIL DO

 BEGIN
 WITH CURRENTPATHSTEP@ DO
```

```
  BEGIN
  OBSTACLESET:=[];
  CURRPOLYHEDRON:=FIRSTOBSTACLE;

  WHILE CURRPOLYHEDRON<>NIL DO

   BEGIN
   WITH CURRPOLYHEDRON@ DO
    IF INRANGE(LOWERBOUND1,UPPERBOUND1,DOFANGLE[1],NORMALBOUNDS1)
    OR INRANGE(LOWERBOUND2,UPPERBOUND2,DOFANGLE[1],NORMALBOUNDS2) THEN
     OBSTACLESET:=OBSTACLESET+[POLYHEDRONNUMBER];
   CURRPOLYHEDRON:=CURRPOLYHEDRON@.NEXTPOLYHEDRON
   END;

  WORKPIECESET:=[];
  CURRWORKPIECE:=FIRSTWORKPIECE;

  WHILE CURRWORKPIECE<>NIL DO

   BEGIN
   WITH CURRWORKPIECE@.BOXPOLYHEDRON@ DO
    CASE HOLDSTATUS OF
    HOLDNEWWORKPIECE,HOLDOLDWORKPIECE:
     IF POLYHEDRONNUMBER<>WKPCNUMBER THEN

      BEGIN
      IF (INRANGE(LOWERBOUND1,UPPERBOUND1,DOFANGLE[1],NORMALBOUNDS1)
      AND (POLYHEDRONNUMBER<>PREVWKPCNUMBER))
      OR INRANGE(LOWERBOUND2,UPPERBOUND2,DOFANGLE[1],NORMALBOUNDS2) THEN
       WORKPIECESET:=WORKPIECESET+[POLYHEDRONNUMBER]
      END;

    NOWORKPIECE:
     IF (INRANGE(LOWERBOUND1,UPPERBOUND1,DOFANGLE[1],NORMALBOUNDS1)
     AND (POLYHEDRONNUMBER<>WKPCNUMBER)
     AND (POLYHEDRONNUMBER<>PREVWKPCNUMBER))
     OR INRANGE(LOWERBOUND2,UPPERBOUND2,DOFANGLE[1],NORMALBOUNDS2) THEN
      WORKPIECESET:=WORKPIECESET+[POLYHEDRONNUMBER]
    END;

   CURRWORKPIECE:=CURRWORKPIECE@.NEXTDEFINEBOX
   END

  END;

 CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR
 END;

CHECKPATH(HOLDSTATUS);

CURRENTPATHSTEP:=PATHSTART;
J:=1;
```

```
CASE HOLDSTATUS OF

HOLDNEWWORKPIECE,HOLDOLDWORKPIECE:
 WHILE CURRENTPATHSTEP<>NIL DO

  BEGIN
  GRAPHICS(CURRENTPATHSTEP,WKPCNUMBER,J);
  J:=J+1;
  CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR
  END;

NOWORKPIECE:
 IF FIRSTPATH THEN

  BEGIN
  WHILE CURRENTPATHSTEP<>PATHGOAL DO

   BEGIN
   GRAPHICS(CURRENTPATHSTEP,0,J);
   J:=J+1;
   CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR
   END;

  GRAPHICS(CURRENTPATHSTEP,WKPCNUMBER,J)
  END

 ELSE
  IF LASTPATH THEN

   BEGIN
   GRAPHICS(CURRENTPATHSTEP,WKPCNUMBER,1);
   J:=2;
   CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR;

   WHILE CURRENTPATHSTEP<>NIL DO

    BEGIN
    GRAPHICS(CURRENTPATHSTEP,0,J);
    J:=J+1;
    CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR
    END

   END

  ELSE

   BEGIN
   GRAPHICS(CURRENTPATHSTEP,PREVWKPCNUMBER,1);
   J:=2;
   CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR;

   WHILE CURRENTPATHSTEP<>PATHGOAL DO

    BEGIN
```

```
    GRAPHICS(CURRENTPATHSTEP,0,J);
    J:=J+1;
    CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR
    END;

   GRAPHICS(CURRENTPATHSTEP,WKPCNUMBER,J)
   END

END;

PATHNUMBER:=PATHNUMBER+1
END;

PROCEDURE DELETEPATH;

PROCEDURE DELETEPOLYHEDRON(VAR POINTER:POINTPOLYHEDRON);

BEGIN
WITH POINTER@ DO

 BEGIN
 CURRPOLYVERTEX:=POLYHEDRONVERTEX;

 WHILE CURRPOLYVERTEX<>NIL DO

  BEGIN
  PREVPOLYVERTEX:=CURRPOLYVERTEX;
  CURRPOLYVERTEX:=CURRPOLYVERTEX@.NEXTVERTEX;
  DISPOSE(PREVPOLYVERTEX)
  END;

 CURRENTPLANE:=FIRSTPLANE;

 WHILE CURRENTPLANE<>NIL DO

  BEGIN
  CURRPLANEVERTEX:=CURRENTPLANE@.PLANEVERTEX;

  WHILE CURRPLANEVERTEX<>NIL DO

   BEGIN
   PREVPLANEVERTEX:=CURRPLANEVERTEX;
   CURRPLANEVERTEX:=CURRPLANEVERTEX@.NEXTPLANEVERTEX;
   DISPOSE(PREVPLANEVERTEX)
   END;

  PREVIOUSPLANE:=CURRENTPLANE;
  CURRENTPLANE:=CURRENTPLANE@.NEXTPLANE;
  DISPOSE(PREVIOUSPLANE)
  END

 END;

DISPOSE(POINTER)
```

```
END;

BEGIN
CURRENTPATHSTEP:=PATHSTART;

WHILE CURRENTPATHSTEP<>PATHGOAL DO

 BEGIN
 WITH CURRENTPATHSTEP@ DO

  BEGIN
  DELETEPOLYHEDRON(BASE@.BOXPOLYHEDRON);
  DISPOSE(BASE);
  DELETEPOLYHEDRON(LINK1@.BOXPOLYHEDRON);
  DISPOSE(LINK1);
  DELETEPOLYHEDRON(LINK2@.BOXPOLYHEDRON);
  DISPOSE(LINK2);
  DELETEPOLYHEDRON(LINK3@.BOXPOLYHEDRON);
  DISPOSE(LINK3);
  DELETEPOLYHEDRON(WRIST@.BOXPOLYHEDRON);
  DISPOSE(WRIST);
  DELETEPOLYHEDRON(FINGER1@.BOXPOLYHEDRON);
  DISPOSE(FINGER1);
  DELETEPOLYHEDRON(FINGER2@.BOXPOLYHEDRON);
  DISPOSE(FINGER2);

  IF WORKPIECE<>NIL THEN

   BEGIN
   DELETEPOLYHEDRON(WORKPIECE@.BOXPOLYHEDRON);
   DISPOSE(WORKPIECE)
   END

  END;

 PREVIOUSPATHSTEP:=CURRENTPATHSTEP;
 CURRENTPATHSTEP:=CURRENTPATHSTEP@.NEXTMANIPULATOR;
 DISPOSE(PREVIOUSPATHSTEP)
 END

END;

                            ****************
                            * MAIN PROGRAM *
                            ****************

BEGIN
REWRITE(ERRORS);
PAGENUMBER:=STARTPAGENUMBER;
AREA(8.5,11);
PLOTS(0,0,0);
MESSAGE:='FINDPATH OUTPUT BEGINS ...';
SYMBOL(2.55,9,0.15,MESSAGE,0,26);
PRINTPAGENUMBER;
```

```
PLOT(0,0,999);

GRAPHICSBASIS[1,1]:=1;
GRAPHICSBASIS[1,2]:=0;
GRAPHICSBASIS[1,3]:=0;
GRAPHICSBASIS[2,1]:=0;
GRAPHICSBASIS[2,2]:=1/SQRT(2);
GRAPHICSBASIS[2,3]:=1/SQRT(2);
GRAPHICSBASIS[3,1]:=0;
GRAPHICSBASIS[3,2]:=-1/SQRT(2);
GRAPHICSBASIS[3,3]:=1/SQRT(2);

INPUTOBSTACLES;
INPUTWORKPIECES;

MANIPULATORERROR:=FALSE;

IF LLINK1<LLINK2 THEN

 BEGIN
 MANIPULATORERROR:=TRUE;
 WRITELN(ERRORS,'LLINK1 < LLINK2 *':17)
 END;

IF (0.9*HBASE-SQRT(SQR(0.5*H1LINK1)+SQR(0.1*LLINK1)))
<SAFETYFACTOR THEN

 BEGIN
 MANIPULATORERROR:=TRUE;
 WRITELN(ERRORS,'BOTTOM CORNER OF LINK1 MAY HIT X-Y PLANE *':42)
 END;

IF (0.5*LBASE)<SQRT(SQR(0.5*H1LINK1)+SQRT(0.1*LLINK1)) THEN

 BEGIN
 MANIPULATORERROR:=TRUE;
 WRITELN(ERRORS,'BOTTOM CORNER OF LINK1 CAN PROTRUDE FROM BASE *':47)
 END;

IF (SQRT(SQR(0.8*LLINK1)+SQR(0.8*LLINK2)-1.28*LLINK1*LLINK2*
(1/SQRT(2)))-SQRT(SQR(0.5*H1LINK1)+SQR(0.1*LLINK1))-SQRT(SQR
(0.9*HLINK3+2*CLEARANCE+HWRIST+2*HFINGER)+SQR(0.5*LWRIST-WFINGER)))
<SAFETYFACTOR THEN

 BEGIN
 MANIPULATORERROR:=TRUE;
 WRITELN(ERRORS,'HELDWORKPIECE MAY HIT BOTTOM CORNER OF LINK1 *':46)
 END;

IF (0.8*LLINK2-SQRT(SQR(0.9*HLINK3+2*CLEARANCE+HWRIST+2*HFINGER)
+SQR(0.5*LWRIST-WFINGER))-SQRT(SQR(0.5*H2LINK1)+SQR(0.1*LLINK1)))
<SAFETYFACTOR THEN

 BEGIN
```

```
 MANIPULATORERROR:=TRUE;
 WRITELN(ERRORS,'HELDWORKPIECE MAY HIT TOP CORNER OF LINK1 *':43)
 END;

IF (0.9*HLINK3+2*CLEARANCE+HWRIST+HFINGER-SQRT(SQR(0.5*H2LINK2)+
SQR(0.1*LLINK2)))<0 THEN

 BEGIN
 MANIPULATORERROR:=TRUE;
 WRITELN(ERRORS,'TOP CORNER OF LINK2 MAY HIT X-Y PLANE *':39)
 END;

IF (SQRT(SQR(0.8*LLINK1)+SQR(0.8*LLINK2)-1.28*LLINK1*LLINK2*
(1/SQRT(2)))-SQRT(SQR(0.5*LBASE)+SQR(0.1*HBASE))-SQRT(SQR
(0.5*H2LINK2)+SQR(0.1*LLINK2)))<SAFETYFACTOR THEN

 BEGIN
 MANIPULATORERROR:=TRUE;
 WRITELN(ERRORS,'TOP CORNER OF LINK2 MAY HIT TOP CORNER OF BASE *':48)
 END;

IF WORKSPACEERROR OR WORKPIECEERROR OR MANIPULATORERROR THEN
 GOTO 1;

RESET(TASK);
FIRSTPATH:=TRUE;
LASTPATH:=FALSE;
PATHNUMBER:=1;
PREVWKPCNUMBER:=0;

WHILE NOT EOF(TASK) DO

 BEGIN
 IF FIRSTPATH THEN

  BEGIN
  FOR I:=1 TO 3 DO

   BEGIN
   READ(TASK,ALPHA[I]);
   ALPHA[I]:=ALPHA[I]*PI/180
   END;

  IF (ALPHA[1]<-PI) OR (ALPHA[1]>PI) THEN

   BEGIN
   WRITELN(ERRORS,'DOFANGLE[1] IS OUT OF RANGE *':29);
   GOTO 1
   END;

  IF ((95*PI/180)>=ALPHA[1]) AND (ALPHA[1]>=(85*PI/180)) THEN

   BEGIN
   WRITELN(ERRORS,'DOFANGLE[1] IS IN DEADBAND *':28);
```

```
      GOTO 1
      END;

     IF (ALPHA[2]<(-80*PI/180)) OR (ALPHA[2]>(80*PI/180)) THEN

      BEGIN
      WRITELN(ERRORS,'DOFANGLE[2] IS OUT OF RANGE *':29);
      GOTO 1
      END;

     IF (ALPHA[3]<0) OR (ALPHA[3]>(135*PI/180)) THEN

      BEGIN
      WRITELN(ERRORS,'DOFANGLE[3] IS OUT OF RANGE *':29);
      GOTO 1
      END;

     READ(TASK,ANGLE);
     ANGLE:=ANGLE*PI/180;
     READ(TASK,DISTANCE);

     IF (DISTANCE<0) OR (DISTANCE>(LWRIST-2*WFINGER)) THEN

      BEGIN
      WRITELN(ERRORS,'JAW OPENING OUT OF RANGE *':26);
      GOTO 1
      END;

     READLN(TASK);
     CONFIGURATION(PATHSTART,FIXED,0,NIL,TRUE,NOWORKPIECE);
     LOCATION:=PATHSTART@.WRIST@.REFERENCE;

     IF LOCATION[3]<(SAFETYFACTOR+HFINGER+CLEARANCE) THEN

      BEGIN
      WRITELN(ERRORS,'FINGER TOO CLOSE TO THE FLOOR *':31);
      GOTO 1
      END;

     IF LOCATION[3]<(LIFTHEIGHT+HFINGER+CLEARANCE) THEN

      BEGIN
      LOCATION[3]:=LIFTHEIGHT+HFINGER+CLEARANCE;
      FINDALPHA;
      CONFIGURATION(POSTPATHSTART,FIXED,0,NIL,TRUE,NOWORKPIECE);
      PATHSTART@.NEXTMANIPULATOR:=POSTPATHSTART;
      PATHPOINTER1:=POSTPATHSTART
      END

     ELSE
      PATHPOINTER1:=PATHSTART
     END

    ELSE
```

```
BEGIN
FOR I:=1 TO 3 DO
 READ(TASK,LOCATION[I]);
LOCATION[3]:=LOCATION[3]+WKPCPOINTER@.HEIGHT+CLEARANCE;
FINDALPHA;
READ(TASK,ANGLE);
ANGLE:=ANGLE*PI/180-PI/2;
CONFIGURATION(PATHGOAL,FIXED,1,WKPCPOINTER,TRUE,HOLDNEWWORKPIECE);
PATHGOAL@.NEXTMANIPULATOR:=NIL;
LOCATION[3]:=LOCATION[3]+LIFTHEIGHT;
FINDALPHA;
CONFIGURATION(PREPATHGOAL,FIXED,1,WKPCPOINTER,TRUE,HOLDNEWWORKPIECE);
PREPATHGOAL@.NEXTMANIPULATOR:=PATHGOAL;
PATHPOINTER2:=PREPATHGOAL;
PRODUCEPATH(WKPCPOINTER,HOLDNEWWORKPIECE);
DELETEPATH;
PATHSTART:=PATHGOAL;

WITH PATHSTART@ DO

 BEGIN
 K:=0;
 ANGLE:=ORIENTATION;
 DISTANCE:=GAP;
 LOCATION:=WRIST@.REFERENCE
 END;

WITH WKPCPOINTER@ DO

 BEGIN
 REFERENCE[1]:=LOCATION[1];
 REFERENCE[2]:=LOCATION[2];
 REFERENCE[3]:=LOCATION[3]-CLEARANCE-HEIGHT;
 THETA:=ANGLE+PI/2
 END;

BOXRECORDS(WKPCPOINTER,FALSE);
PREVWKPCNUMBER:=WKPCNUMBER;
PREVWKPCPOINTER:=WKPCPOINTER;
LOCATION[3]:=LOCATION[3]+HFINGER+CLEARANCE+SAFETYFACTOR;
FINDALPHA;
DISTANCE:=DISTANCE+SAFETYFACTOR;
CONFIGURATION(POSTPATHSTART,FIXED,0,NIL,TRUE,NOWORKPIECE);
PATHSTART@.NEXTMANIPULATOR:=POSTPATHSTART;
PATHPOINTER1:=POSTPATHSTART;
CODE:=' ';
WHILE CODE=' ' DO
 READ(TASK,CODE);
READLN(TASK);

IF CODE='S' THEN

 BEGIN
```

```
LASTPATH:=TRUE;
FOR I:=1 TO 3 DO

 BEGIN
 READ(TASK,ALPHA[I]);
 ALPHA[I]:=ALPHA[I]*PI/180
 END;

IF (ALPHA[1]<-PI) OR (ALPHA[1]>PI) THEN

 BEGIN
 WRITELN(ERRORS,'DOFANGLE[1] WILL BE OUT OF RANGE *':34);
 GOTO 1
 END;

IF ((95*PI/180)>=ALPHA[1]) AND (ALPHA[1]>=(85*PI/180)) THEN

 BEGIN
 WRITELN(ERRORS,'DOFANGLE[1] WILL BE IN DEADBAND *':33);
 GOTO 1
 END;

IF (ALPHA[2]<(-80*PI/180)) OR (ALPHA[2]>(80*PI/180)) THEN

 BEGIN
 WRITELN(ERRORS,'DOFANGLE[2] WILL BE OUT OF RANGE *':34);
 GOTO 1
 END;

IF (ALPHA[3]<0) OR (ALPHA[3]>(135*PI/180)) THEN

 BEGIN
 WRITELN(ERRORS,'DOFANGLE[3] WILL BE OUT OF RANGE *':34);
 GOTO 1
 END;

READ(TASK,ANGLE);
ANGLE:=ANGLE*PI/180;
READ(TASK,DISTANCE);

IF (DISTANCE<=0) OR (DISTANCE>=(LWRIST-2*WFINGER)) THEN

 BEGIN
 WRITELN(ERRORS,'JAW OPENING WILL BE OUT OF RANGE *':34);
 GOTO 1
 END;

READLN(TASK);
CONFIGURATION(PATHGOAL,FIXED,1,NIL,TRUE,NOWORKPIECE);
LOCATION:=PATHGOAL@.WRIST@.REFERENCE;

IF LOCATION[3]<(SAFETYFACTOR+HFINGER+CLEARANCE) THEN

 BEGIN
```

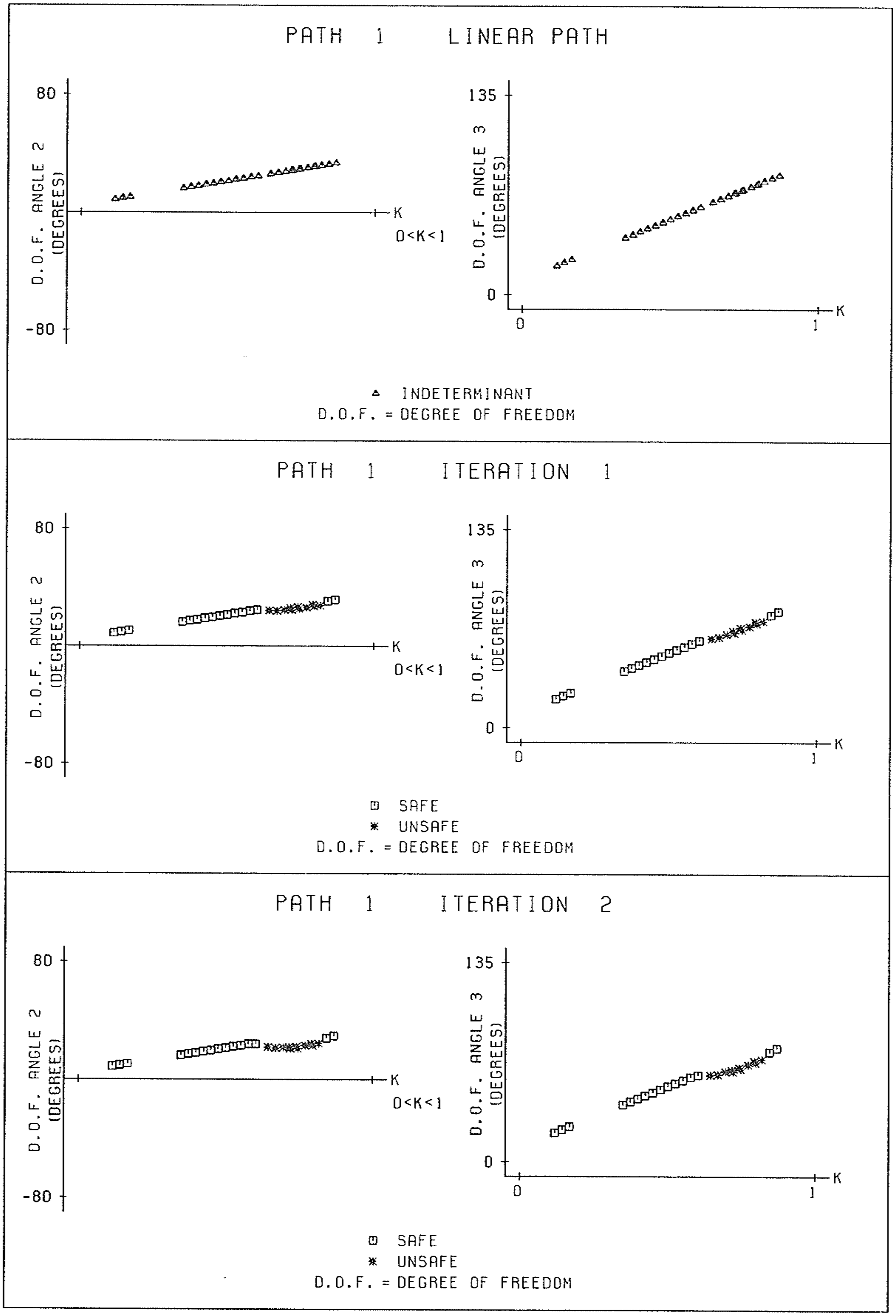
```
    WRITELN(ERRORS,'FINGER WILL BE TOO CLOSE TO':27,
            ' THE FLOOR *':12);
    GOTO 1
    END;

   PATHGOAL@.NEXTMANIPULATOR:=NIL;
   PATHPOINTER2:=PATHGOAL;
   WKPCNUMBER:=0;
   PRODUCEPATH(NIL,NOWORKPIECE)
   END

  END;

 IF NOT EOF(TASK) THEN

  BEGIN
  READLN(TASK,WKPCNUMBER);
  CURRWORKPIECE:=FIRSTWORKPIECE;

  WHILE CURRWORKPIECE<>NIL DO
   IF CURRWORKPIECE@.BOXPOLYHEDRON@.POLYHEDRONNUMBER=WKPCNUMBER
   THEN

    BEGIN
    WITH CURRWORKPIECE@ DO

     BEGIN
     LOCATION[1]:=REFERENCE[1];
     LOCATION[2]:=REFERENCE[2];
     LOCATION[3]:=REFERENCE[3]+HEIGHT+CLEARANCE;
     FINDALPHA;
     ANGLE:=THETA-PI/2;
     DISTANCE:=WIDTH+2*CLEARANCE;
     WKPCPOINTER:=CURRWORKPIECE
     END;

    CURRWORKPIECE:=NIL
    END

   ELSE
    CURRWORKPIECE:=CURRWORKPIECE@.NEXTDEFINEBOX;

  CONFIGURATION(PATHGOAL,FIXED,1,WKPCPOINTER,TRUE,HOLDNEWWORKPIECE);
  PATHGOAL@.NEXTMANIPULATOR:=NIL;
  LOCATION:=PATHGOAL@.WRIST@.REFERENCE;
  LOCATION[3]:=LOCATION[3]+CLEARANCE+HFINGER+SAFETYFACTOR;
  FINDALPHA;
  DISTANCE:=DISTANCE+SAFETYFACTOR;
  CONFIGURATION(PREPATHGOAL,FIXED,1,NIL,TRUE,NOWORKPIECE);
  PREPATHGOAL@.NEXTMANIPULATOR:=PATHGOAL;
  PATHPOINTER2:=PREPATHGOAL;
  PRODUCEPATH(NIL,NOWORKPIECE);
  DELETEPATH;
  PATHSTART:=PATHGOAL;
```

```
  WITH PATHSTART@ DO

   BEGIN
   K:=0;
   ANGLE:=ORIENTATION;
   DISTANCE:=GAP;
   LOCATION:=WRIST@.REFERENCE
   END;

  LOCATION[3]:=LOCATION[3]+CLEARANCE+HFINGER+LIFTHEIGHT;
  FINDALPHA;
  CONFIGURATION(POSTPATHSTART,FIXED,0,WKPCPOINTER,TRUE,
                HOLDNEWWORKPIECE);
  PATHSTART@.NEXTMANIPULATOR:=POSTPATHSTART;
  PATHPOINTER1:=POSTPATHSTART;
  FIRSTPATH:=FALSE
  END

 END;

GOTO 2;
1:MESSAGE:='SEE DA=PROCCA.ERRORS';
  SYMBOL(3,9,0.15,MESSAGE,0,20);
  PRINTPAGENUMBER;
  PLOT(0,0,999);
2:MESSAGE:='... FINDPATH OUTPUT ENDS';
  SYMBOL(2.7,9,0.15,MESSAGE,0,24);
  PRINTPAGENUMBER;
  PLOT(0,0,9999)
END.
```

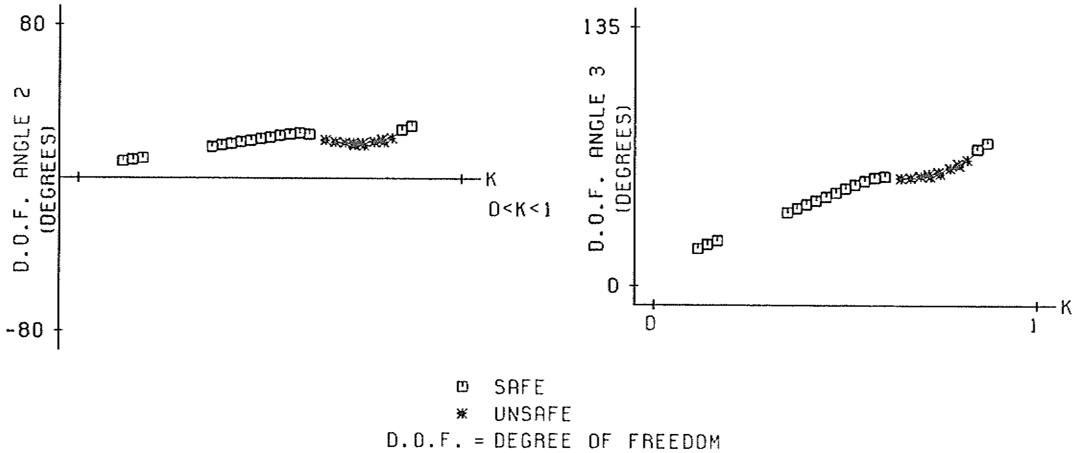
APPENDIX B

Output of Program FINDPATH

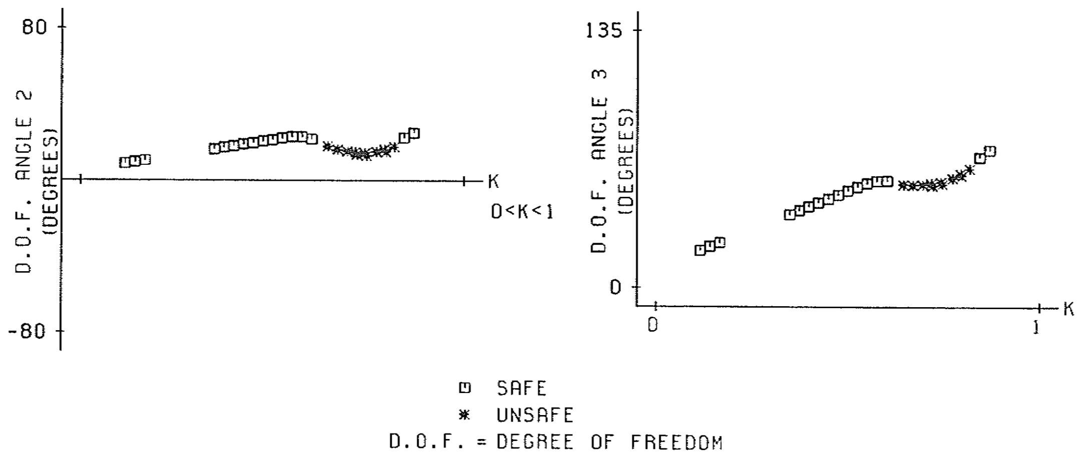
FINDPATH OUTPUT BEGINS ...


PATH 1 LINEAR PATH
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
INDETERMINANT
D.O.F. = DEGREE OF FREEDOM
PATH 1 ITERATION 1
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM
PATH 1 ITERATION 2
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

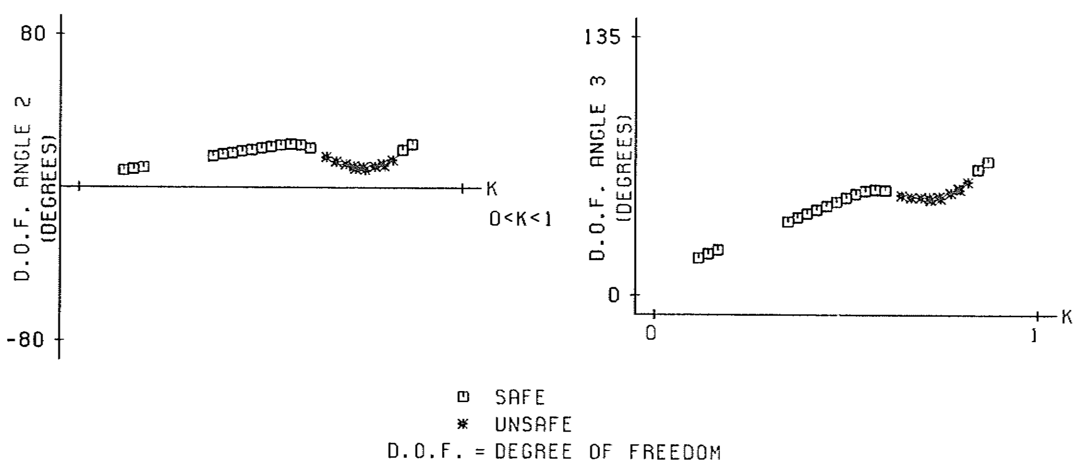
PATH 1 ITERATION 3

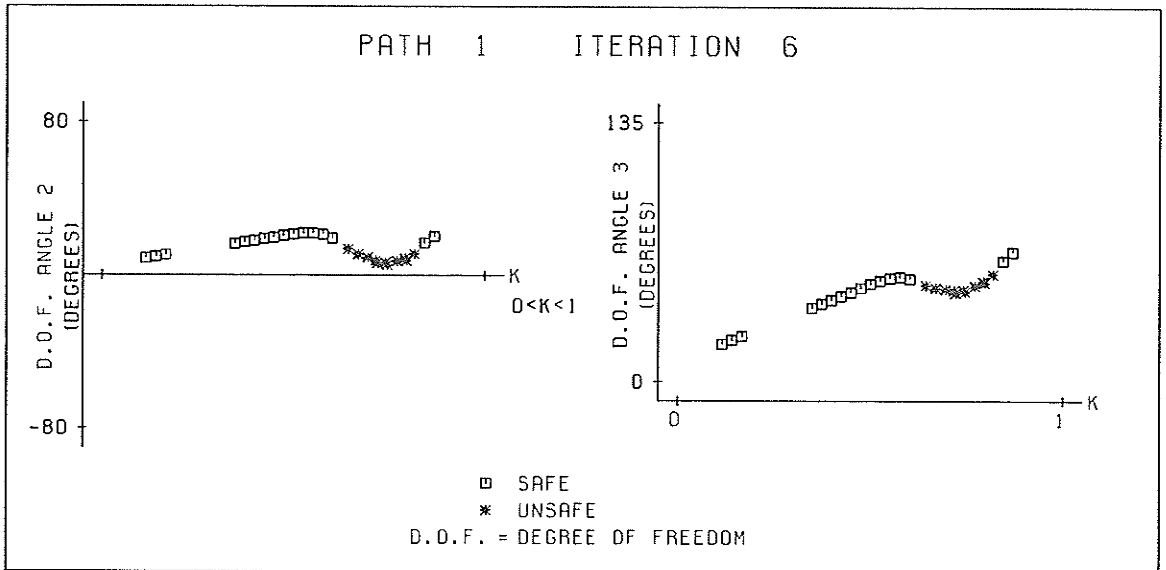


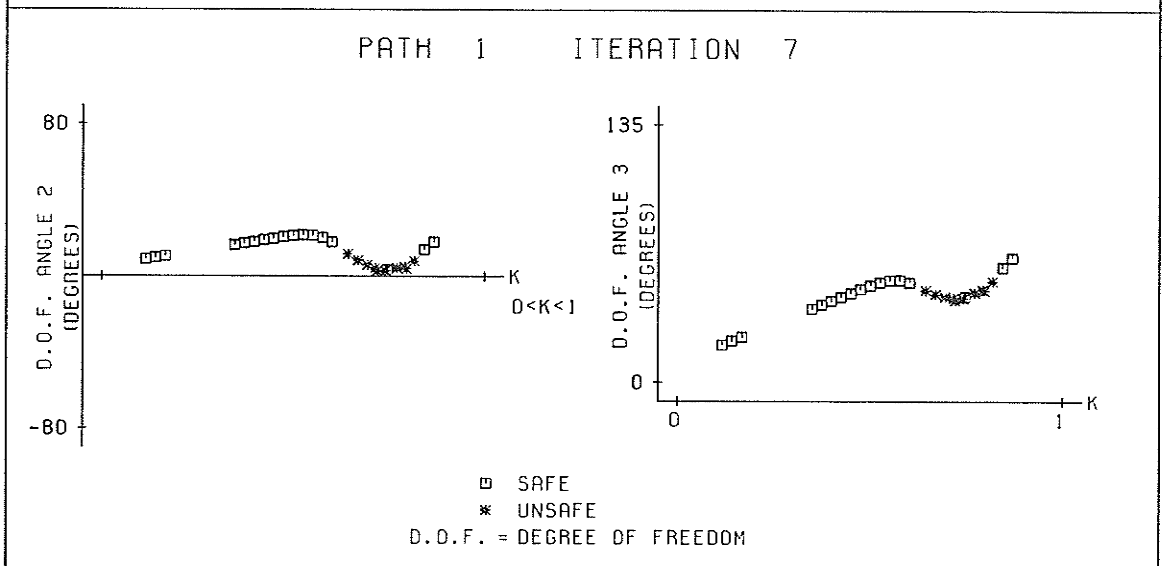
PATH 1 ITERATION 4

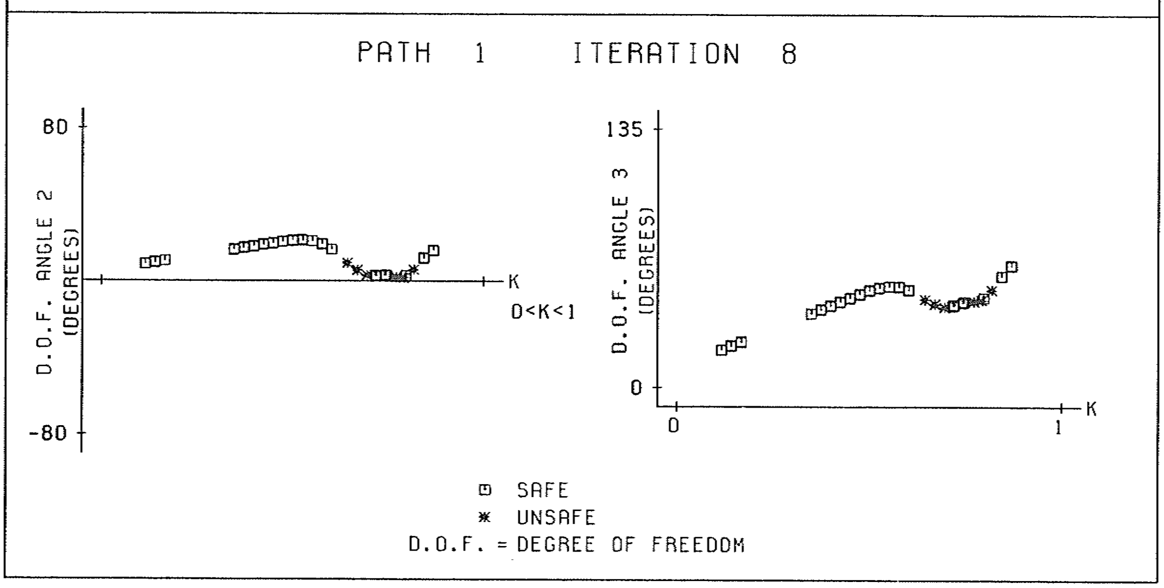


PATH 1 ITERATION 5

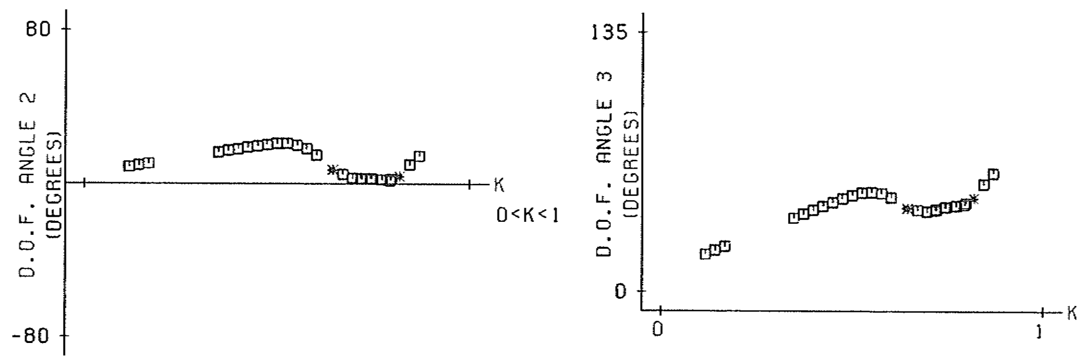



PATH 1 ITERATION 6
80
-80
D.O.F. ANGLE 2
(DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3
(DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM


PATH 1 ITERATION 7
80
-80
D.O.F. ANGLE 2
(DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3
(DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

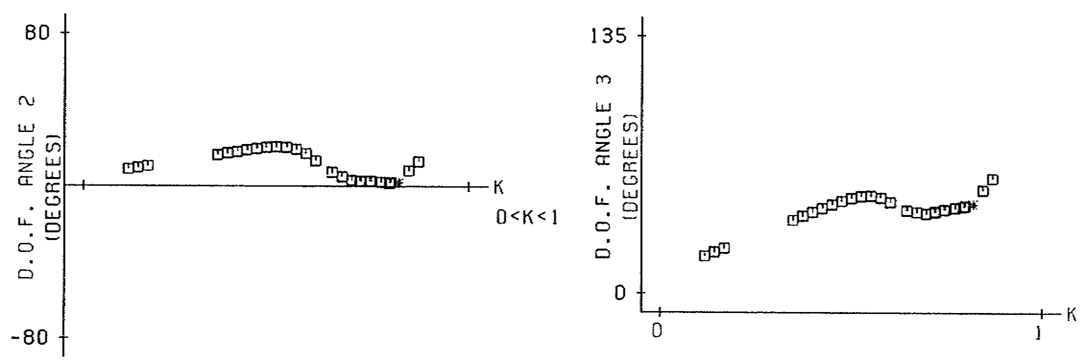

PATH 1 ITERATION 8
80
-80
D.O.F. ANGLE 2
(DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3
(DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

PATH 1 ITERATION 9



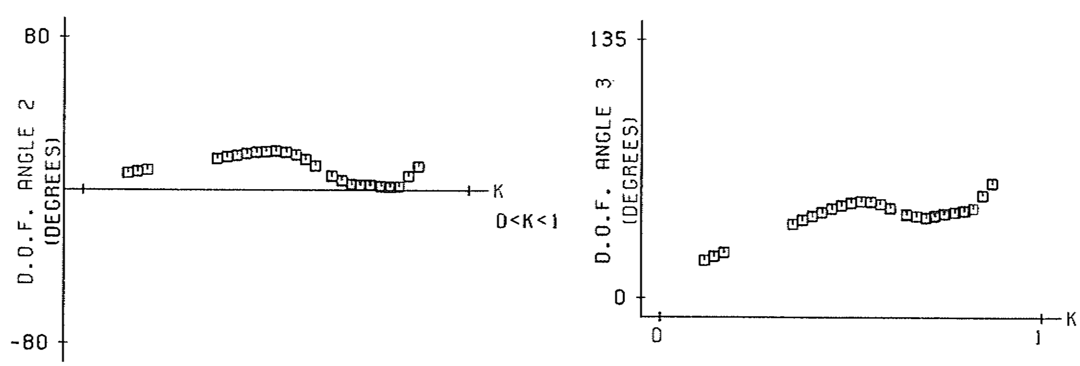
⊡ SAFE
✱ UNSAFE
D.O.F. = DEGREE OF FREEDOM

PATH 1 ITERATION 10

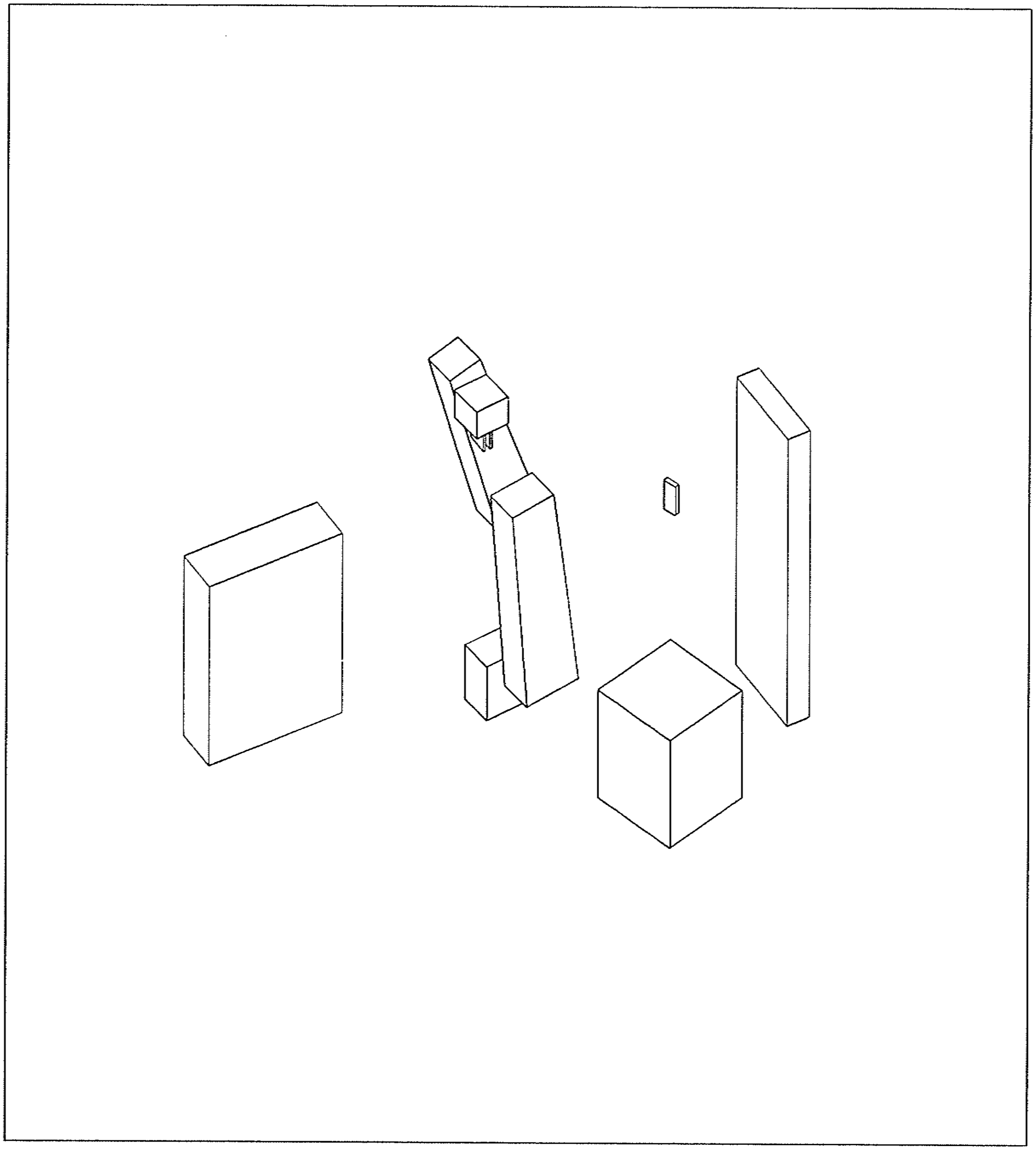


⊡ SAFE
✱ UNSAFE
D.O.F. = DEGREE OF FREEDOM

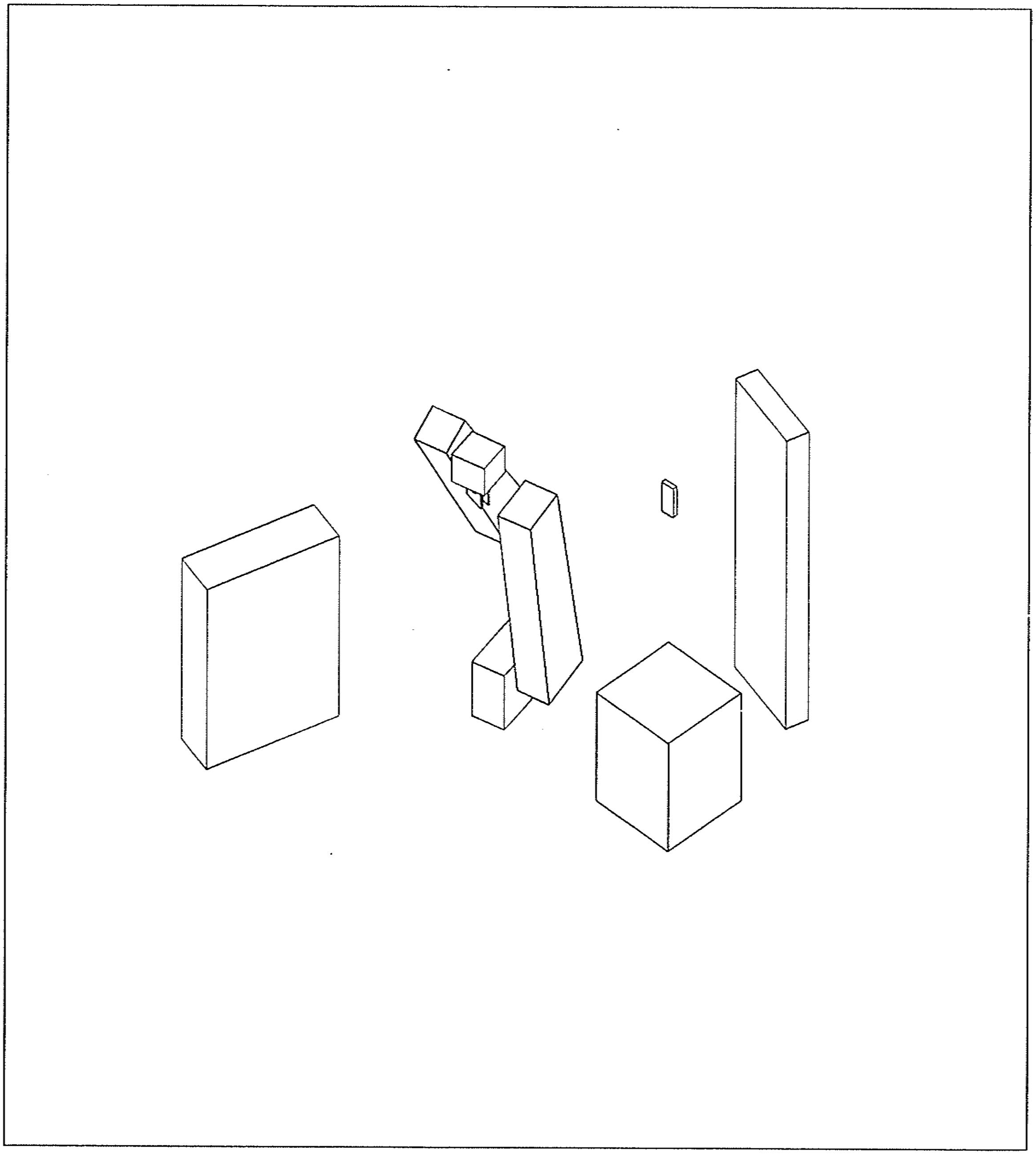
PATH 1 ITERATION 11



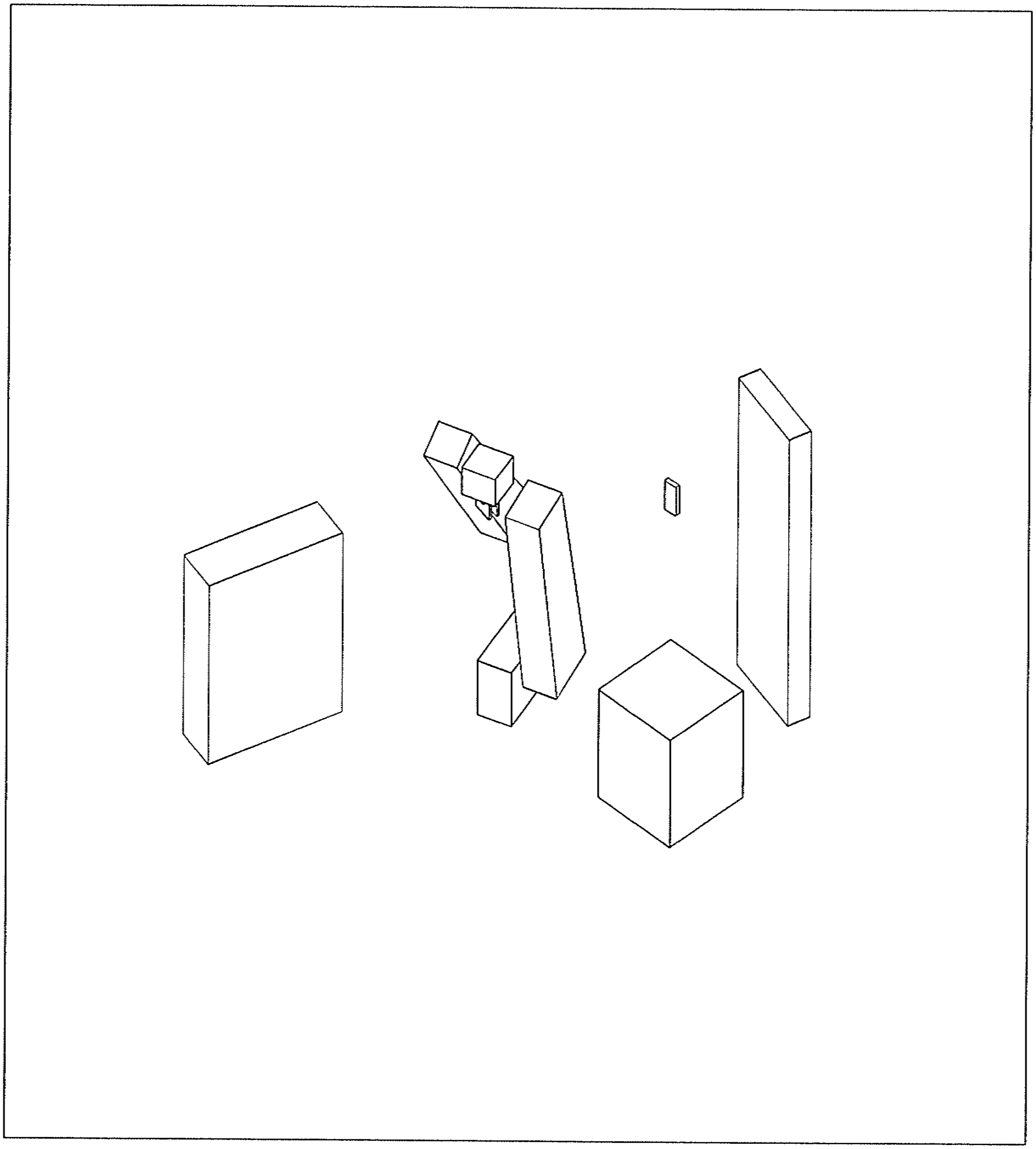
⊡ SAFE
✱ UNSAFE
D.O.F. = DEGREE OF FREEDOM



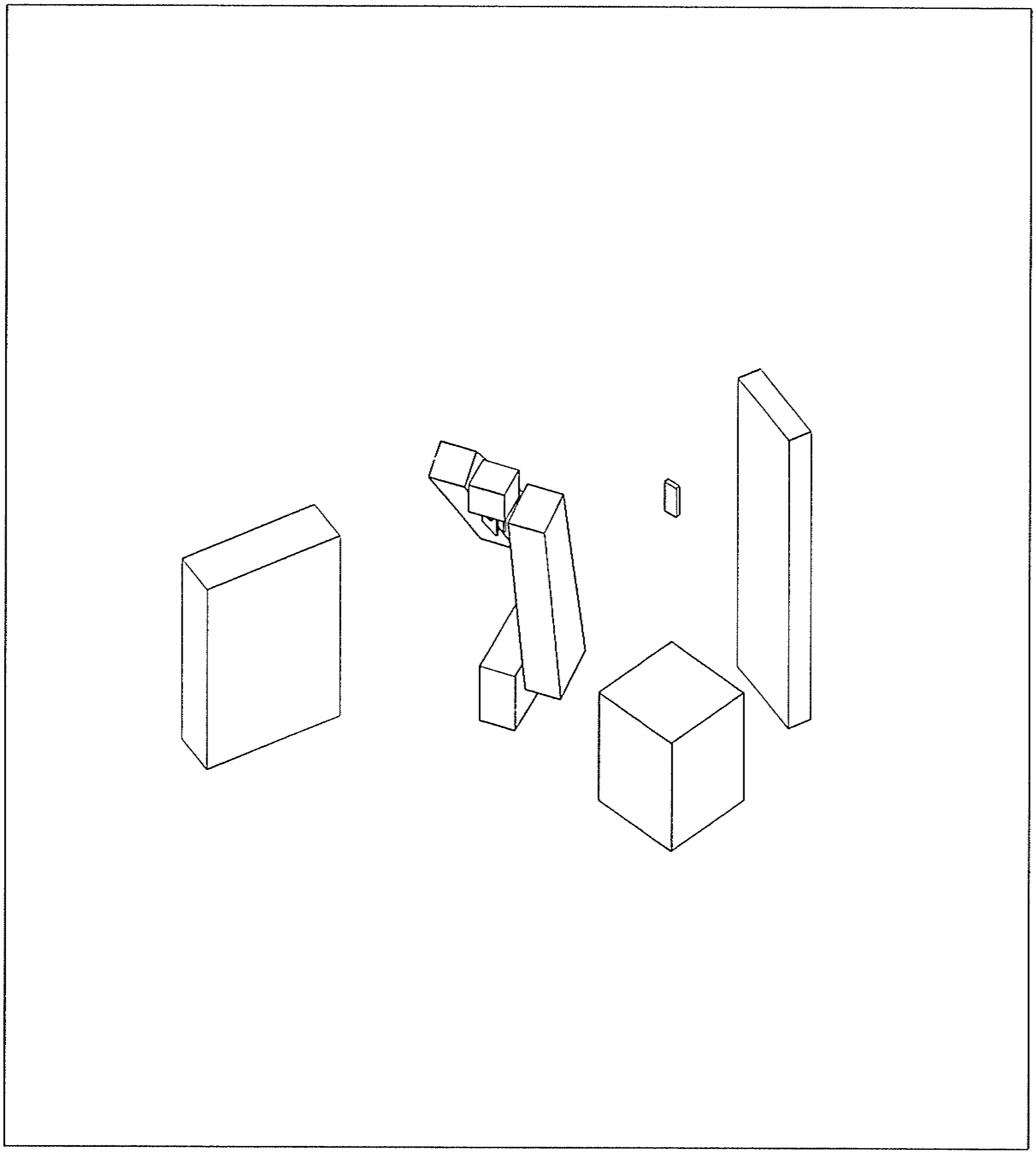
PATH 1 STEP 1
SAFE FIXED CONFIGURATION
PATH IS SAFE



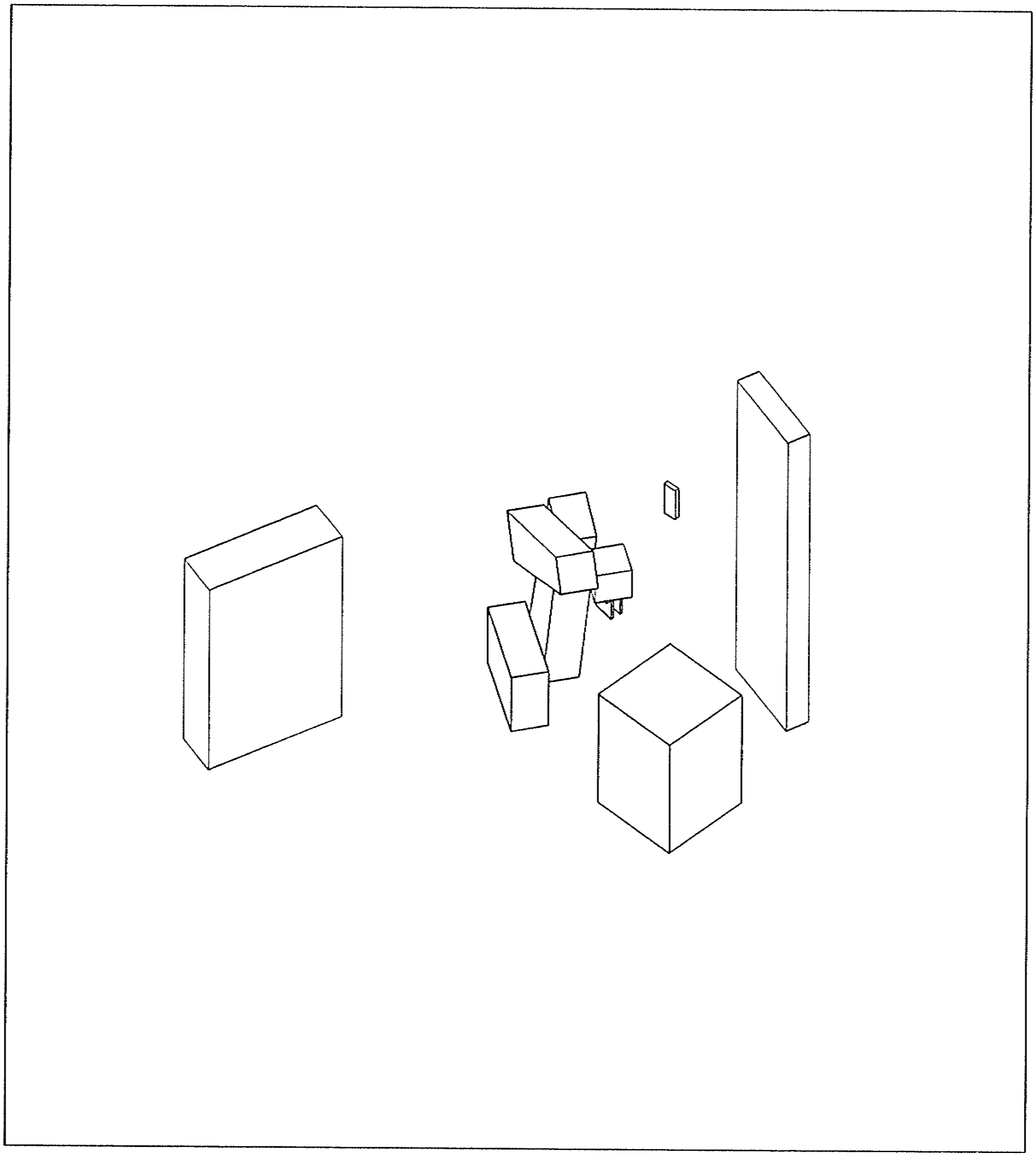
PATH 1 STEP 2
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



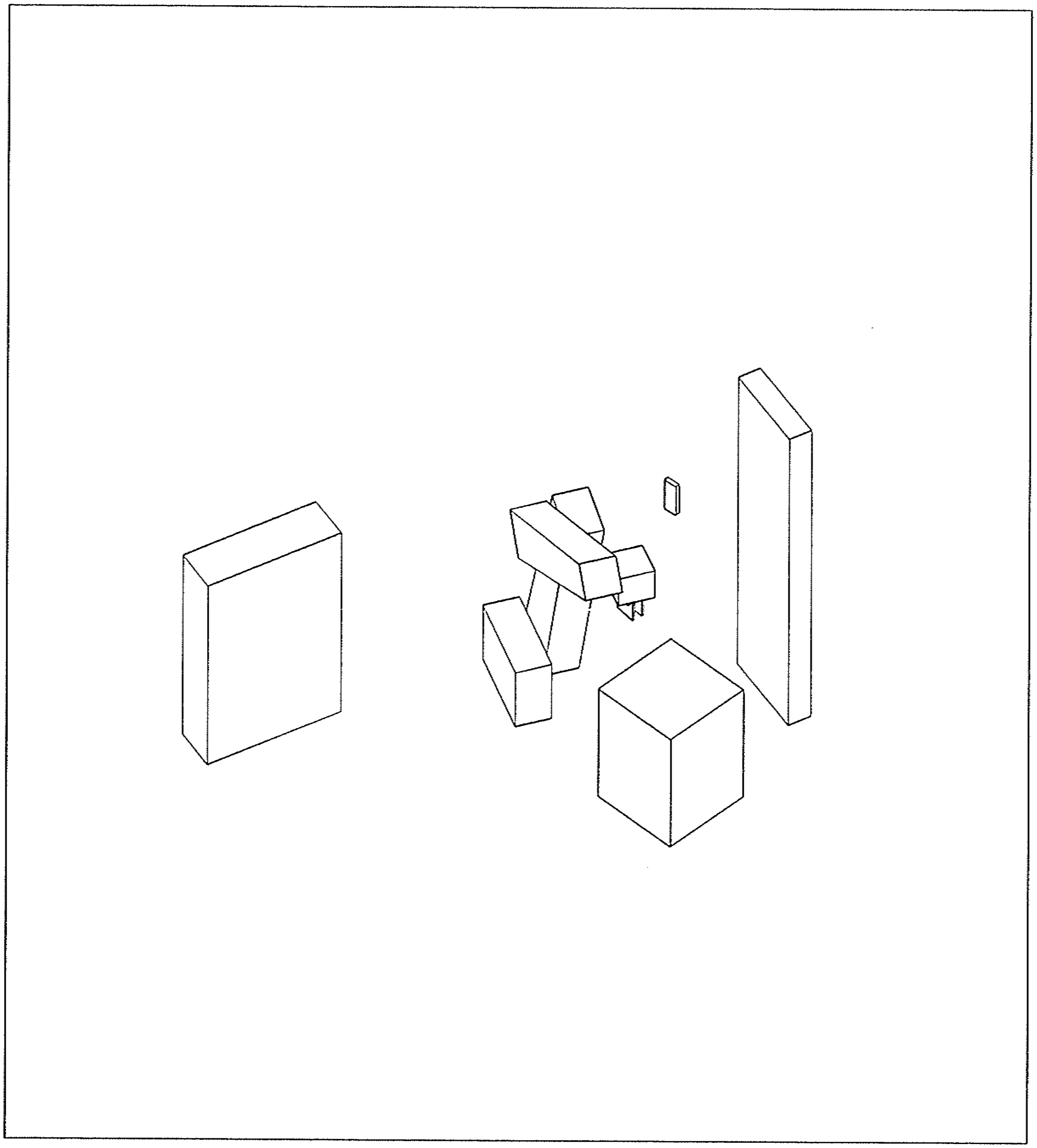
PATH 1 STEP 3
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



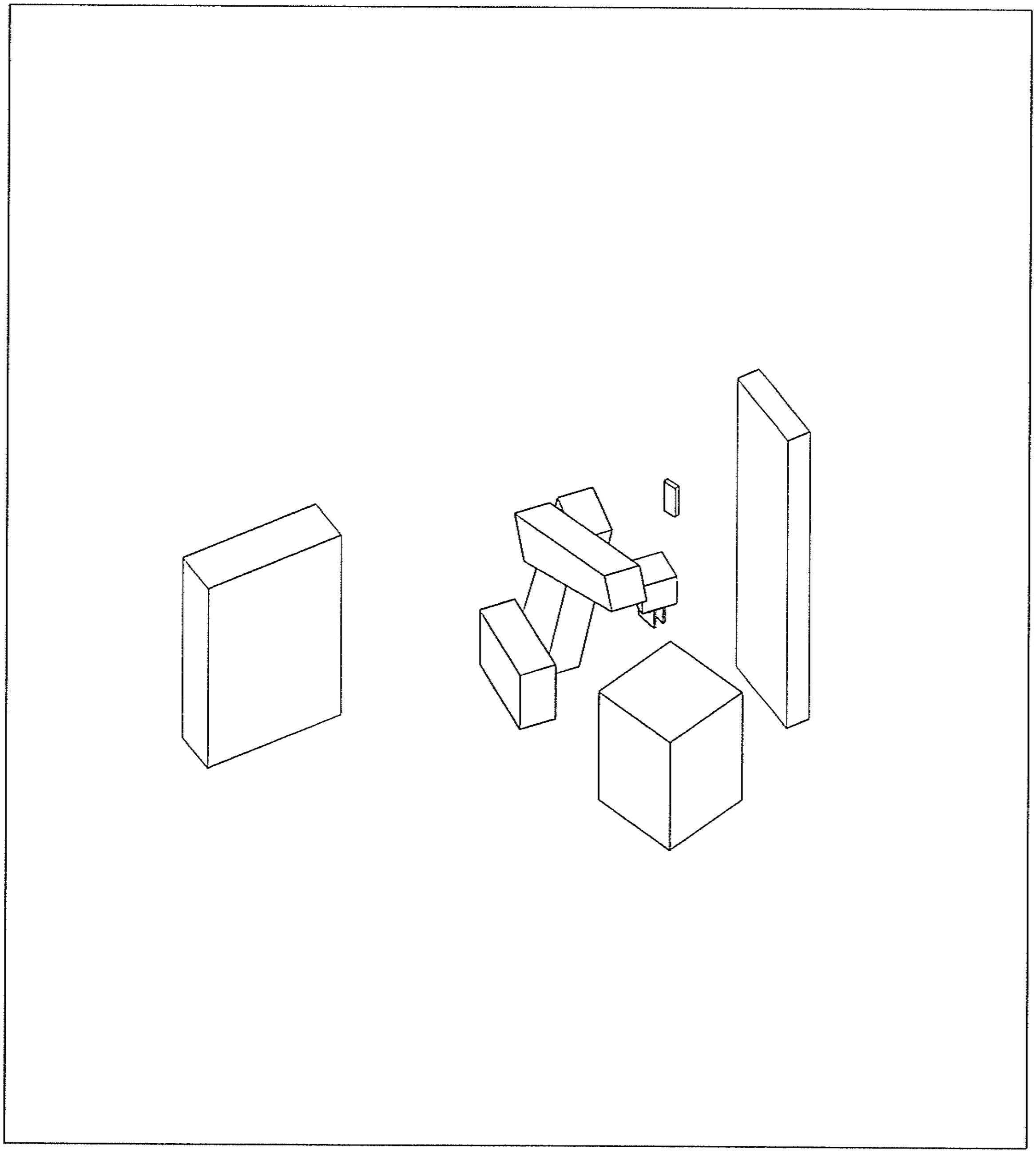
PATH 1 STEP 4
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



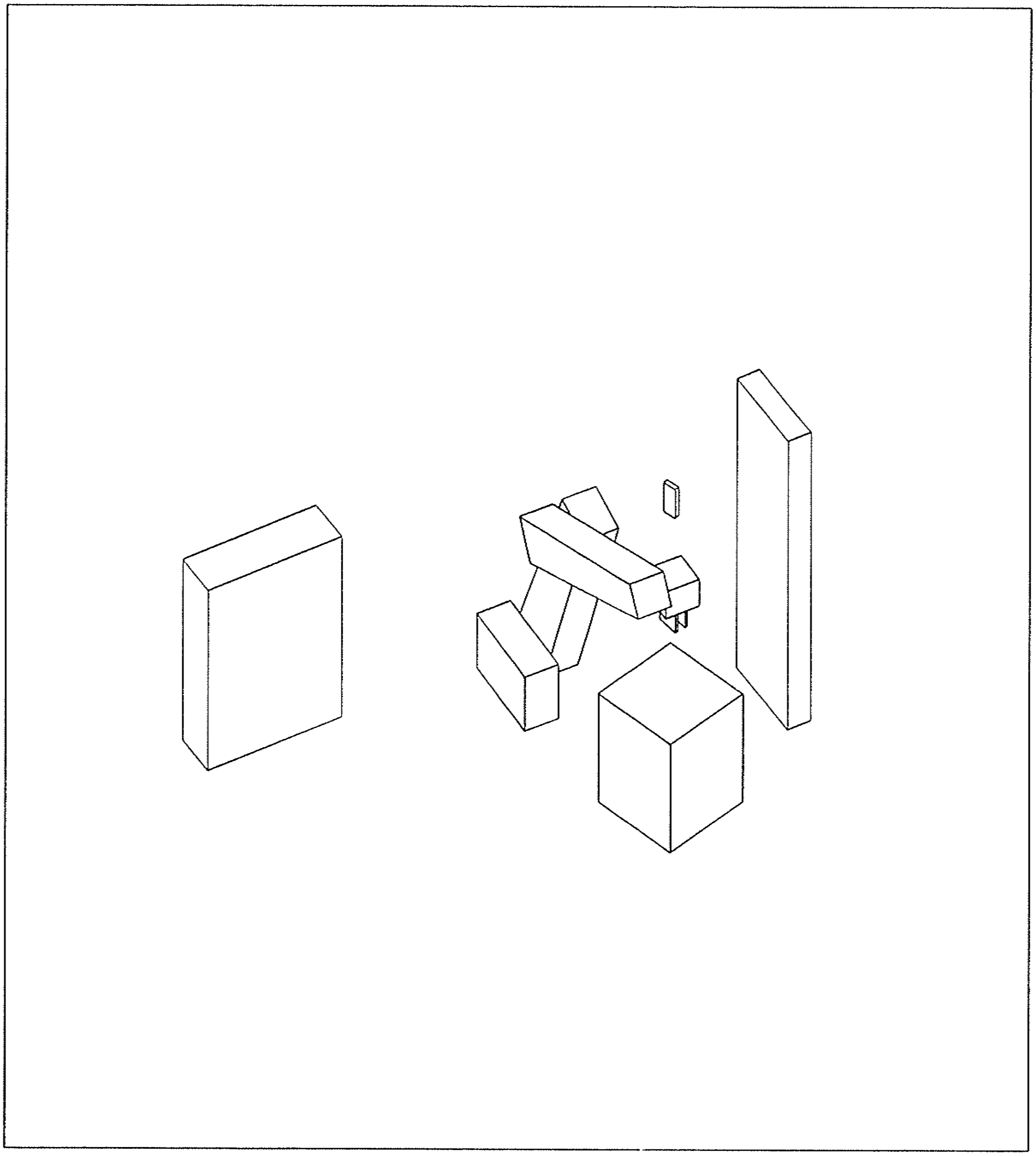
PATH 1 STEP 5
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



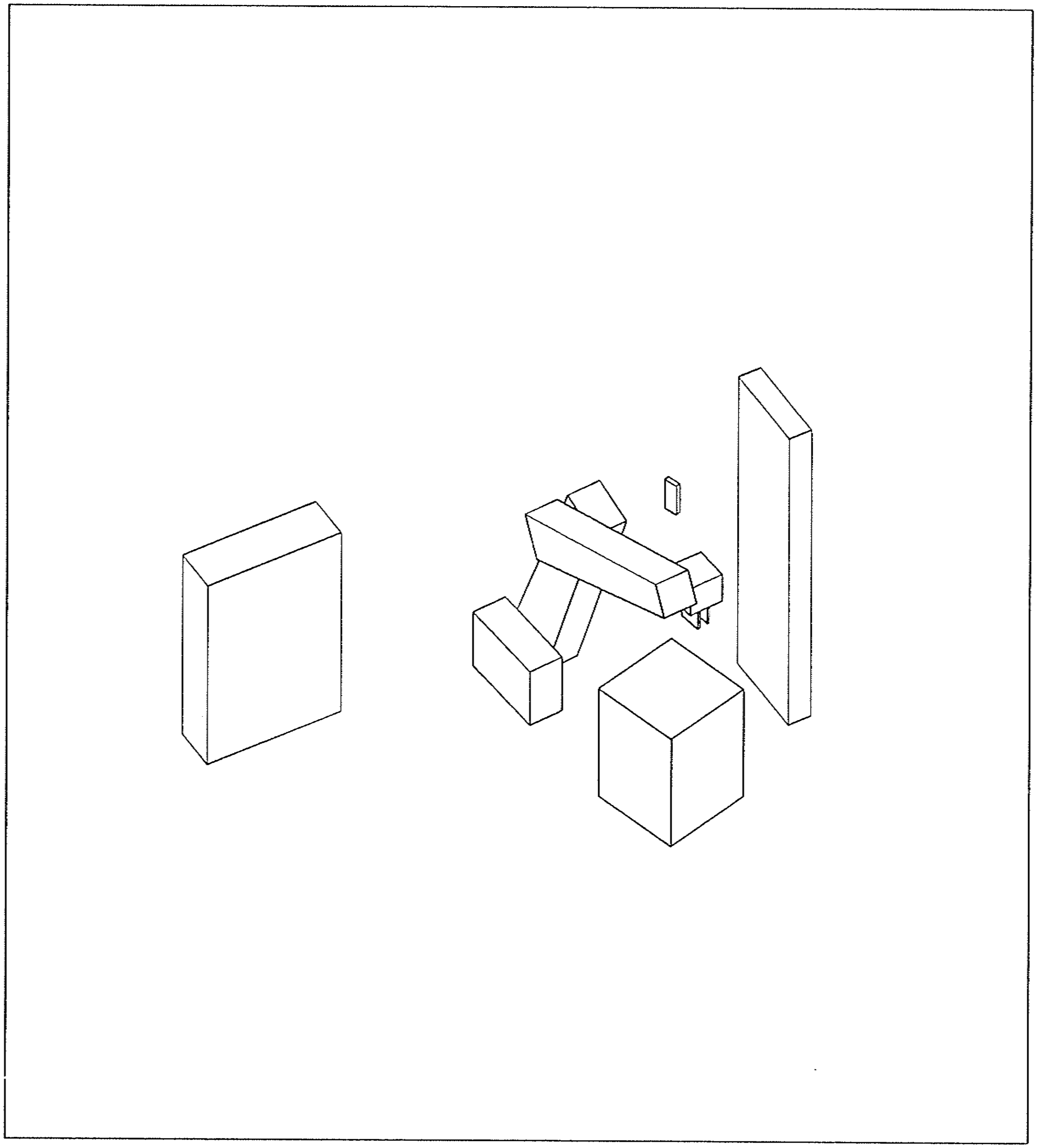
PATH 1 STEP 6
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



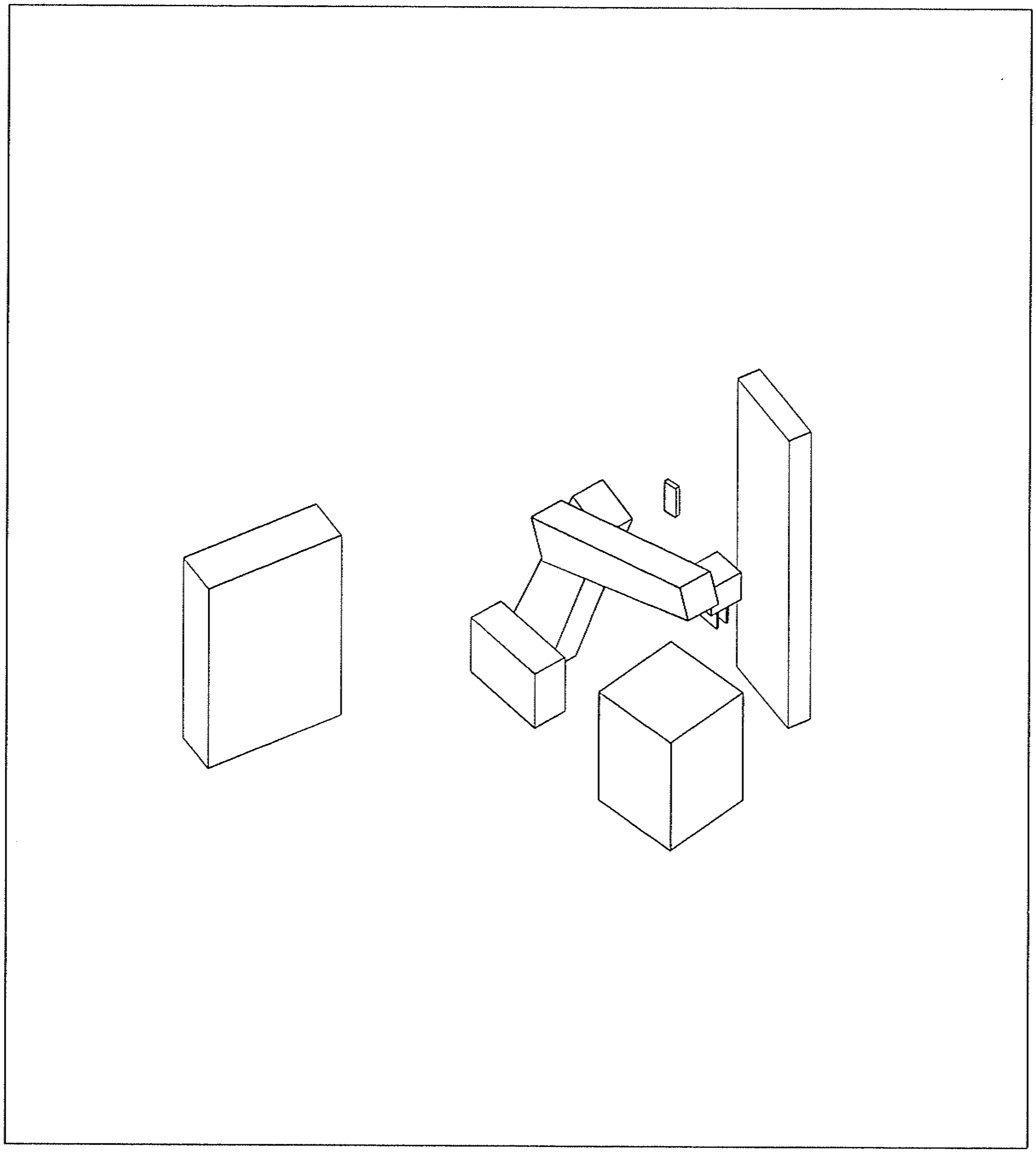
PATH 1 STEP 7
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



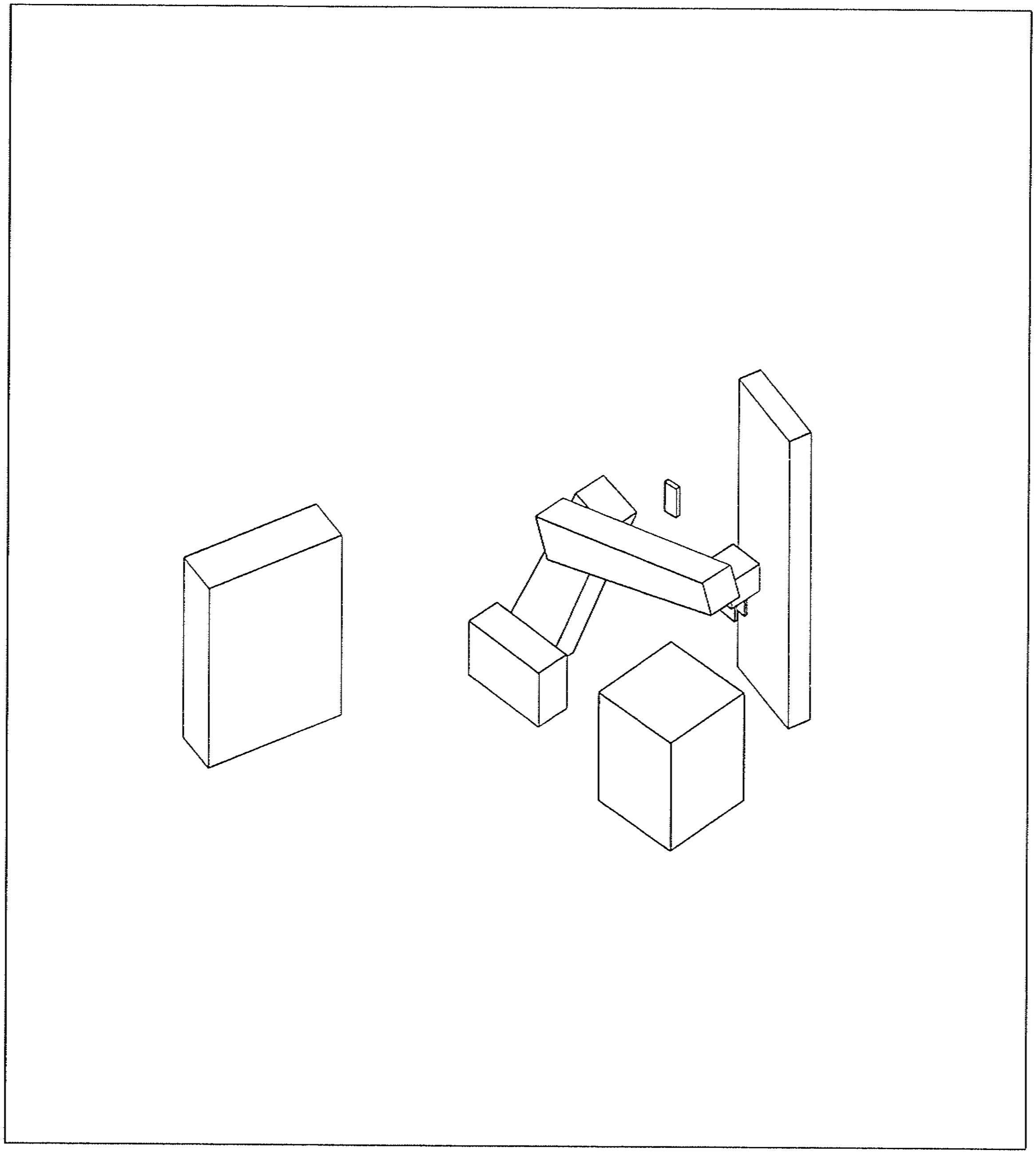
PATH 1 STEP 8
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



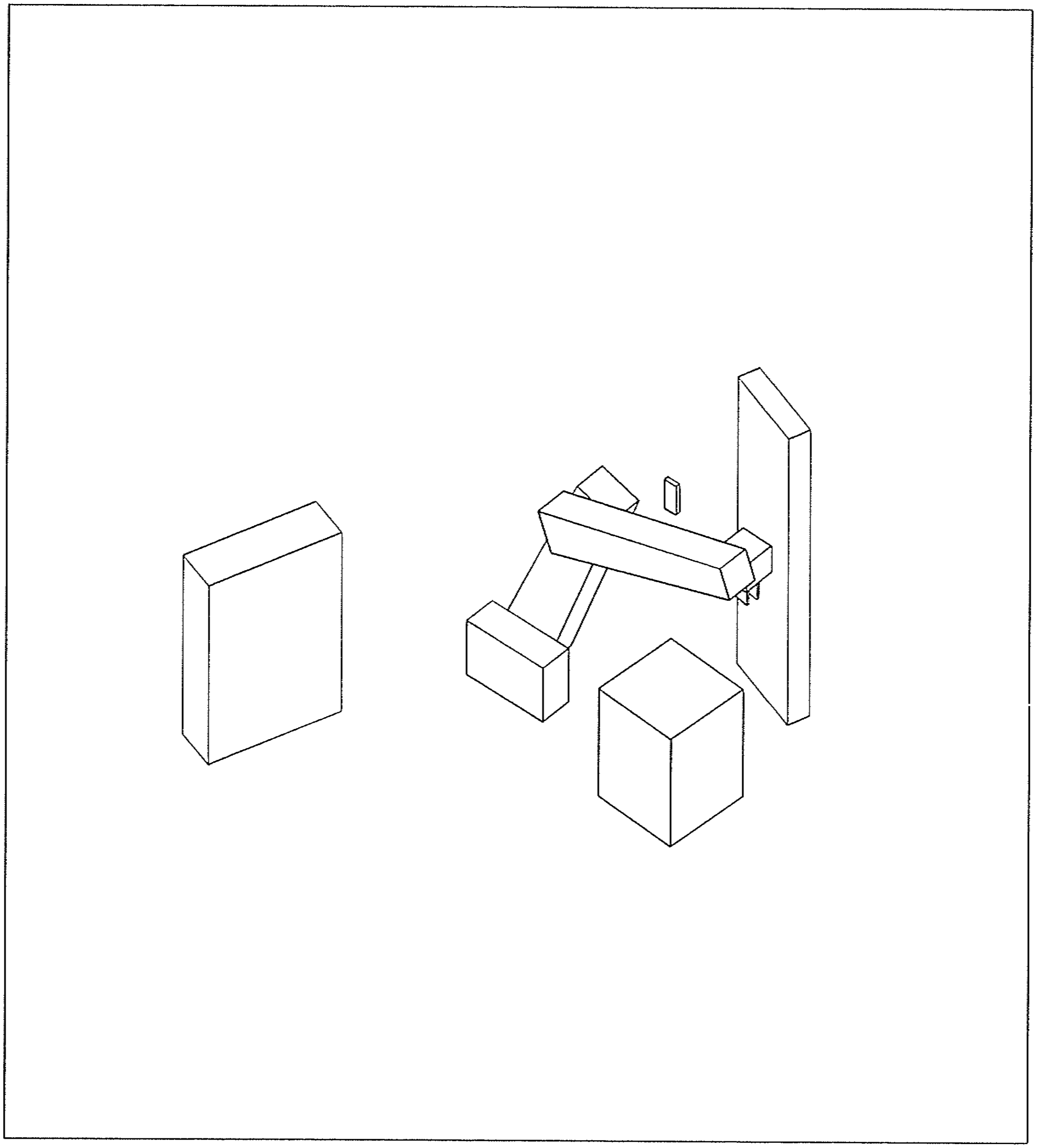
PATH 1 STEP 9
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



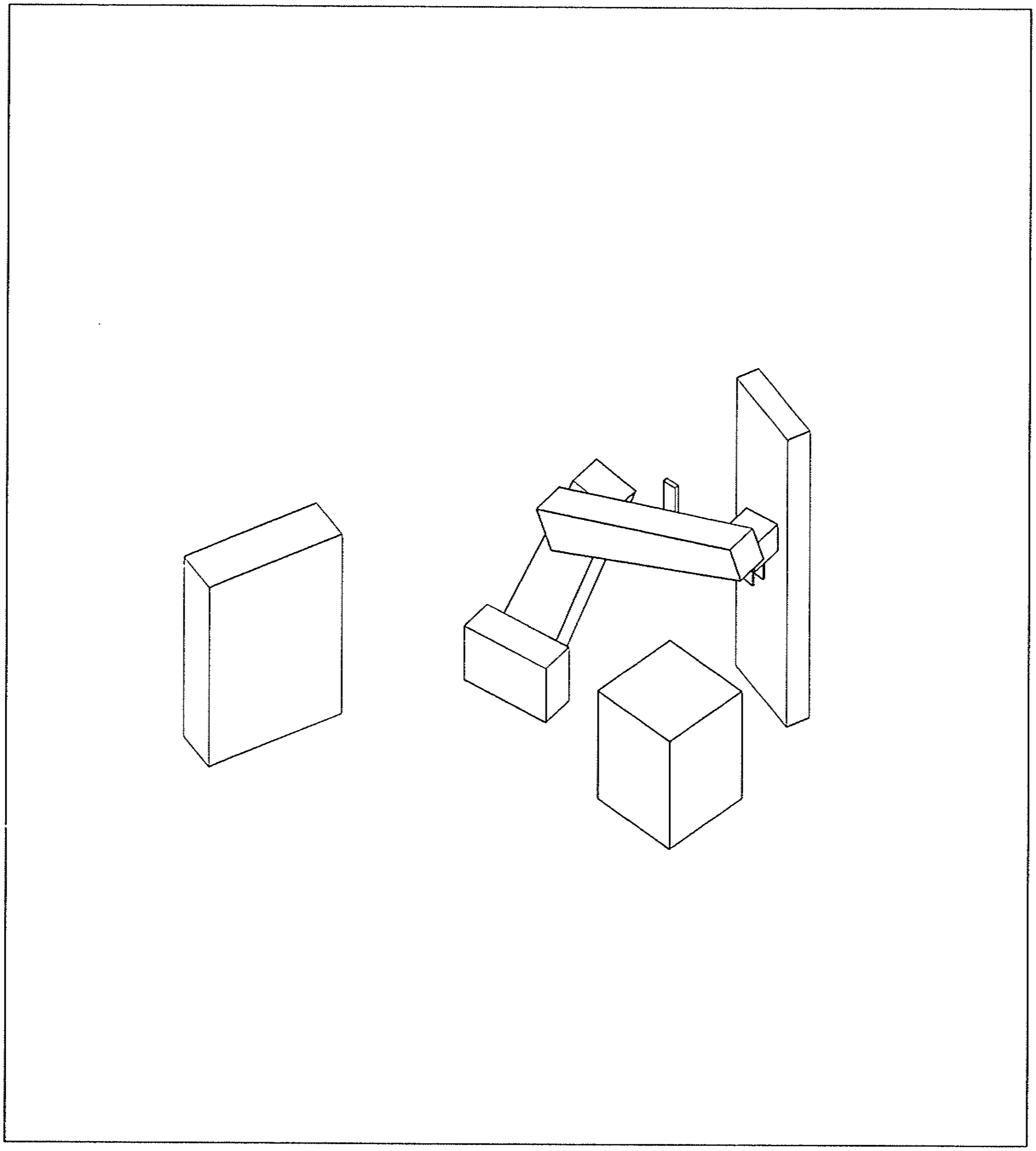
PATH 1 STEP 10
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



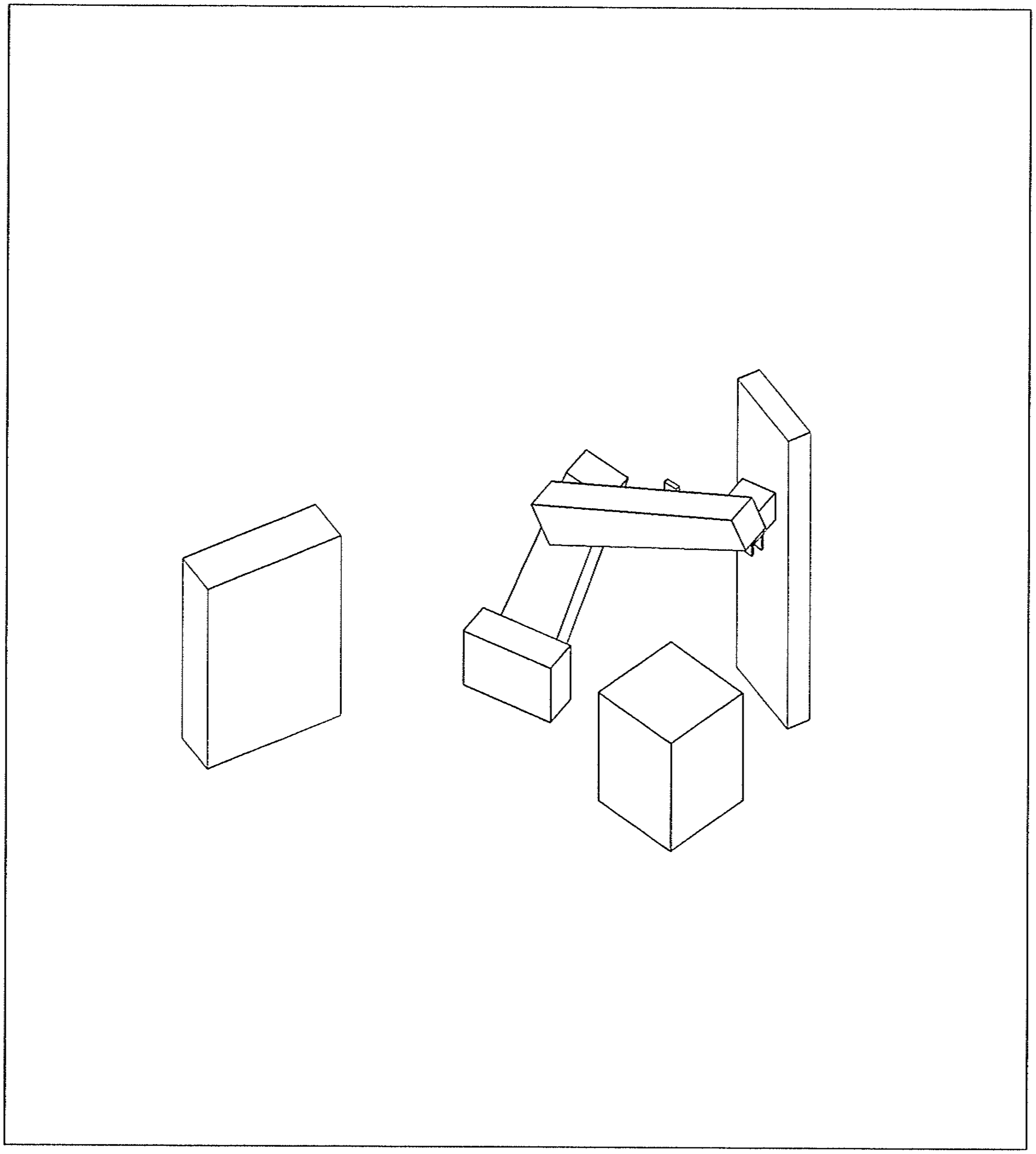
PATH 1 STEP 11
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



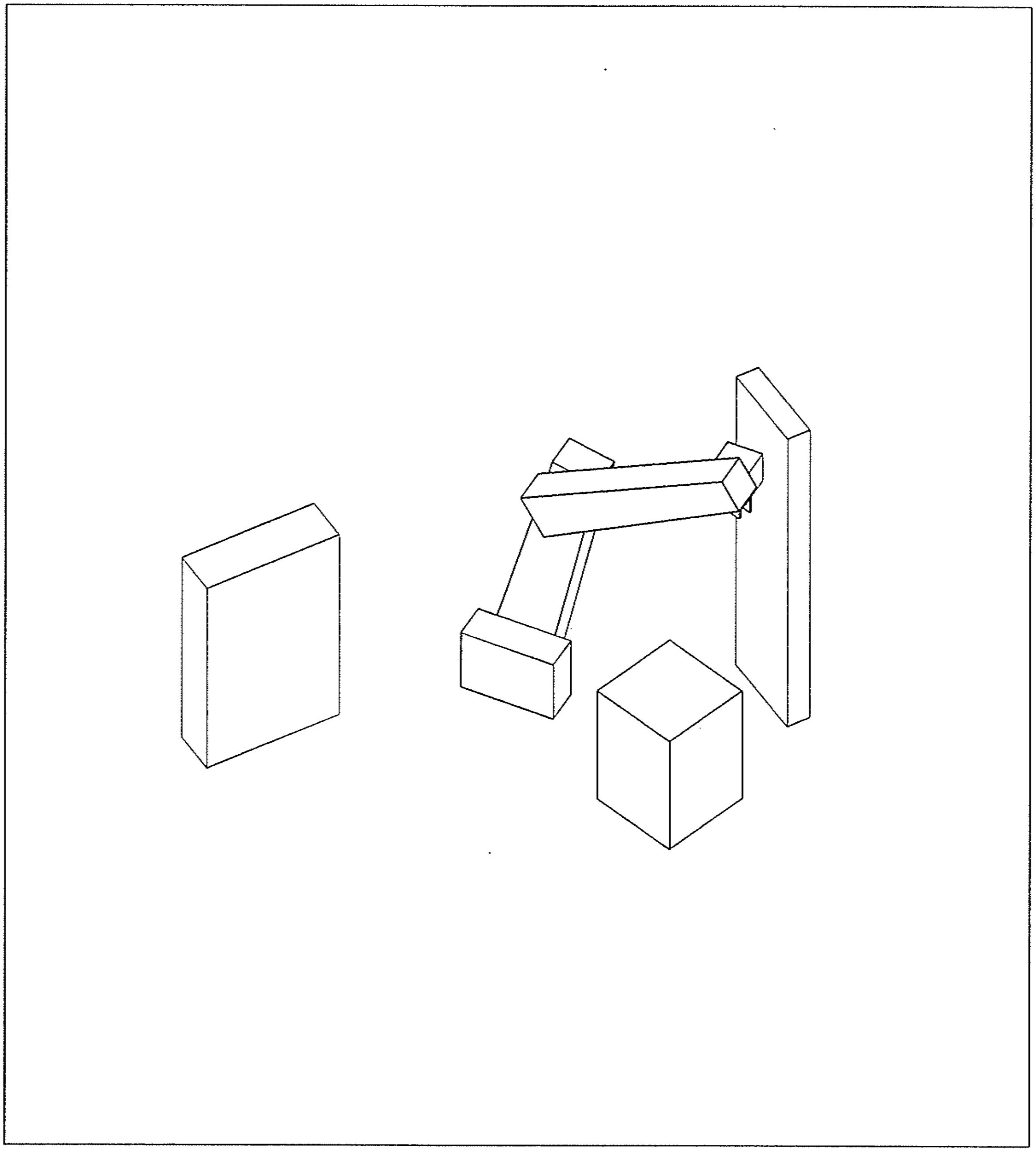
PATH 1 STEP 12
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



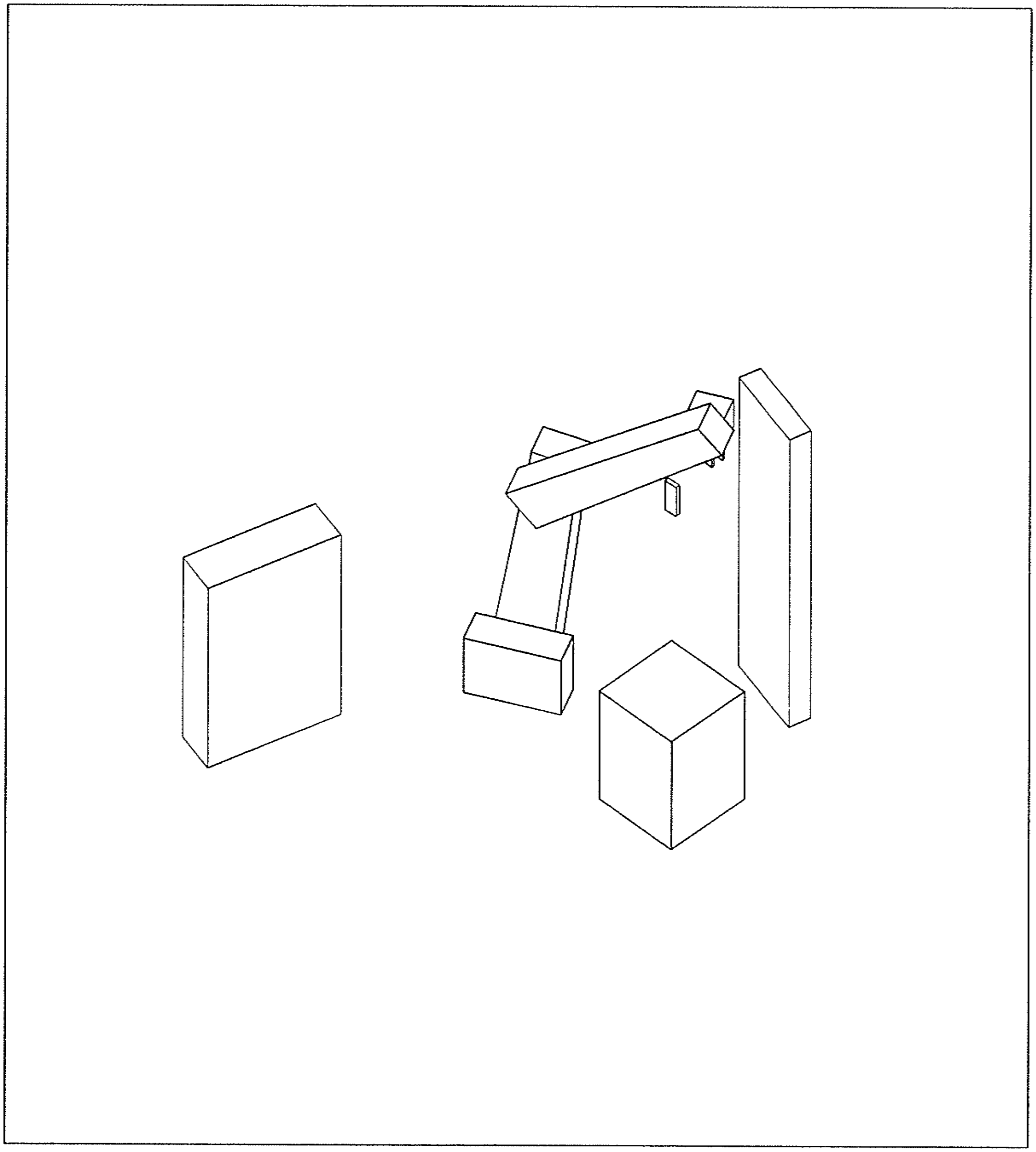
PATH 1 STEP 13
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



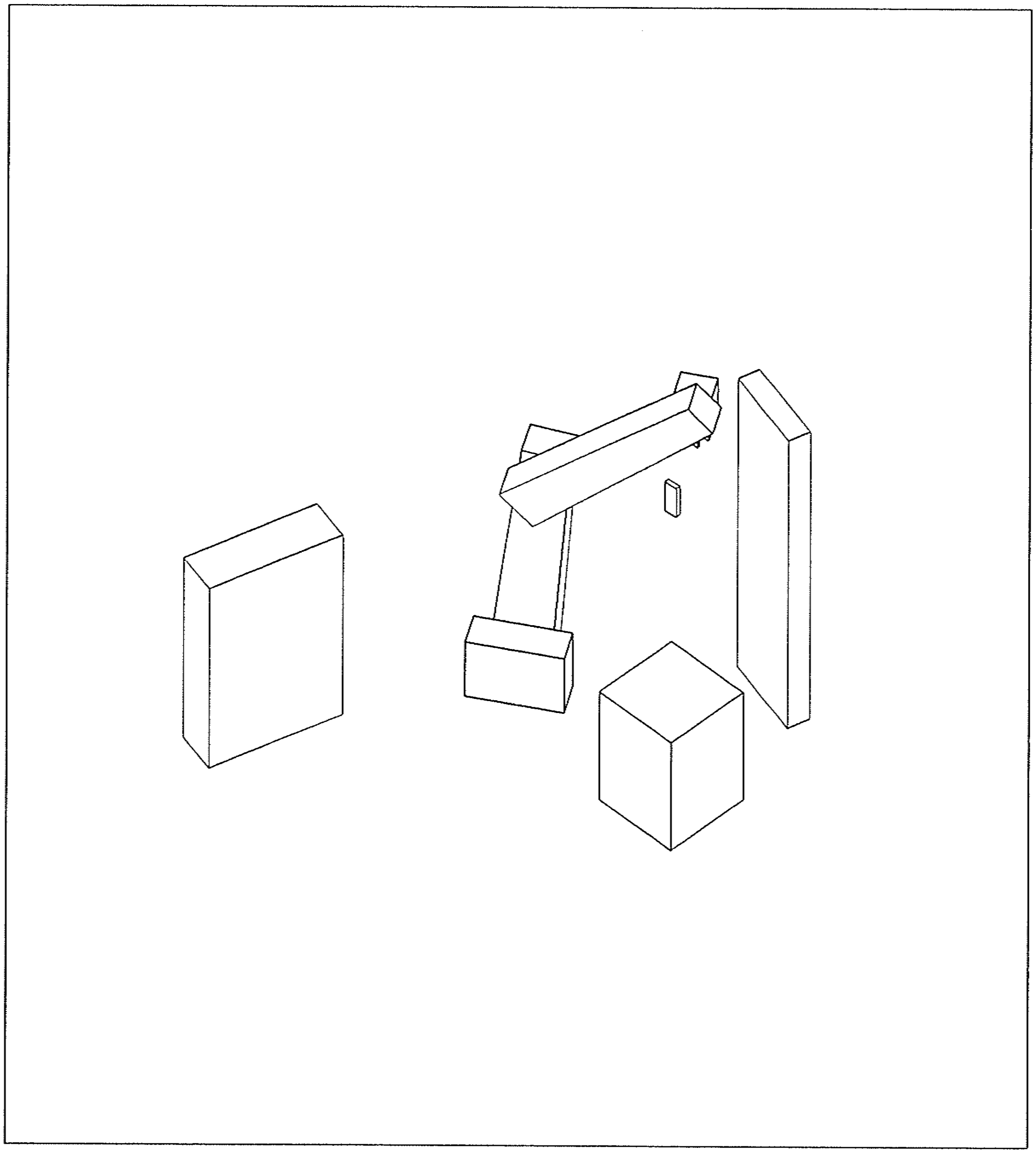
PATH 1 STEP 14
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



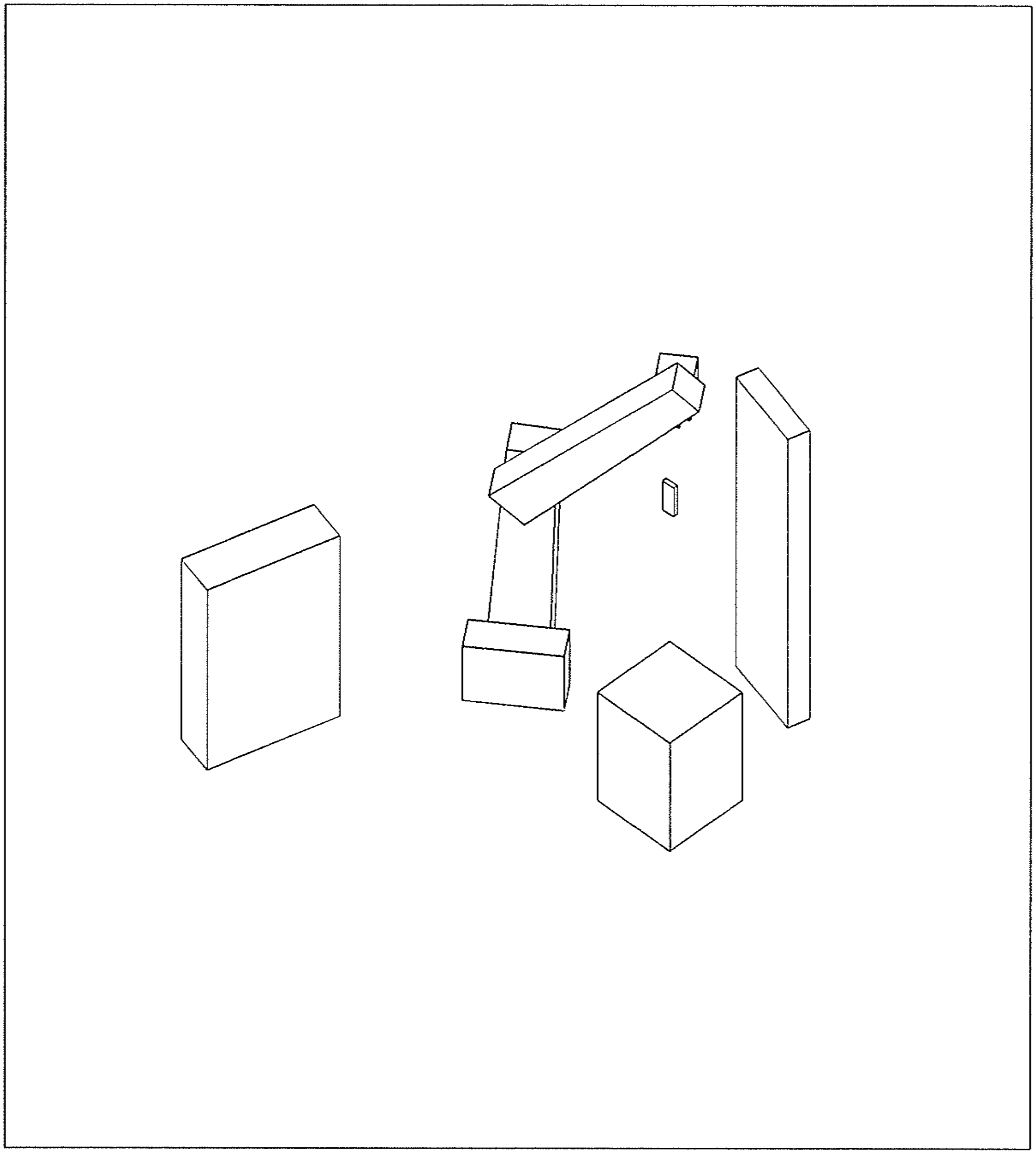
PATH 1 STEP 15
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



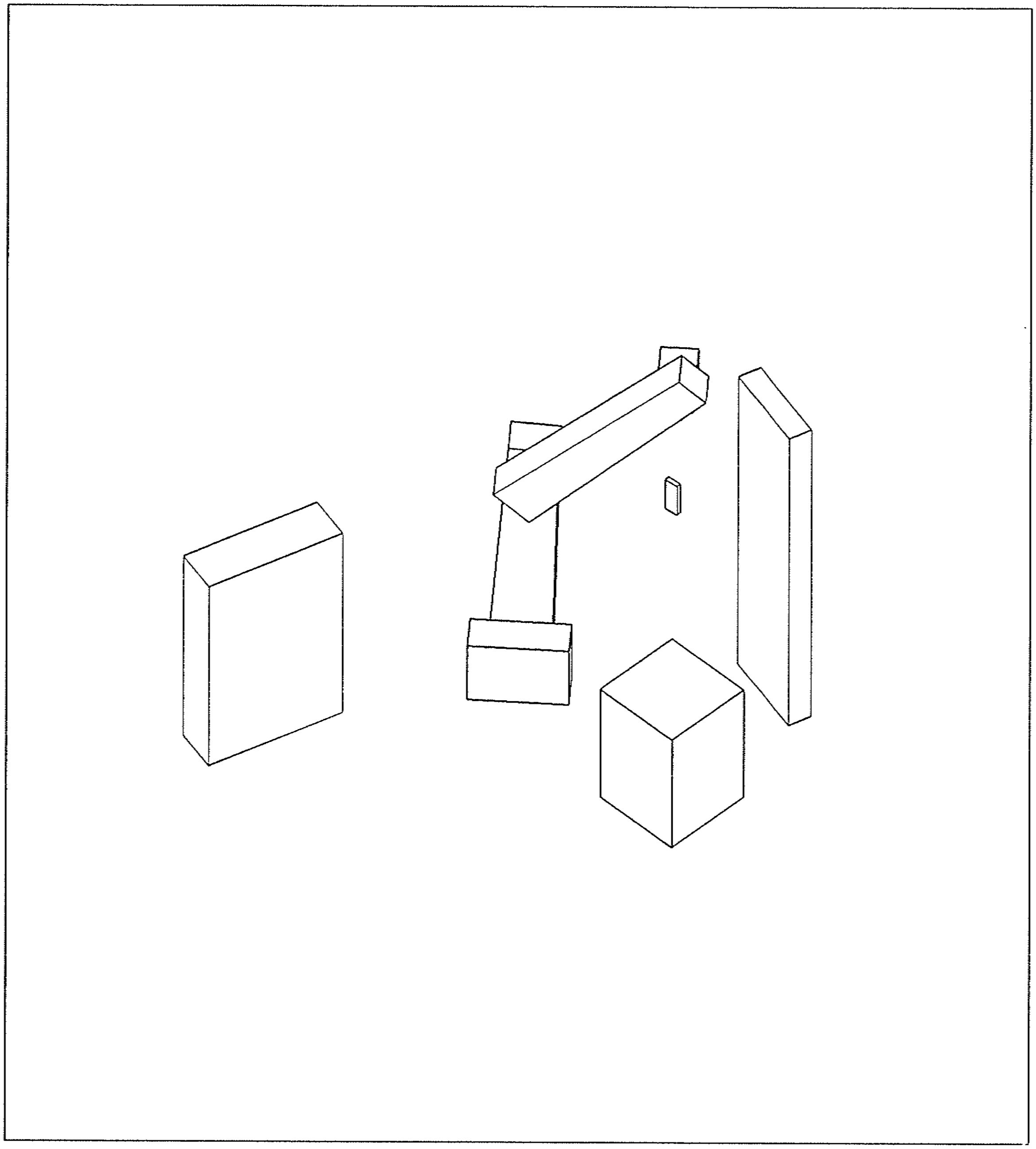
PATH 1 STEP 16
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



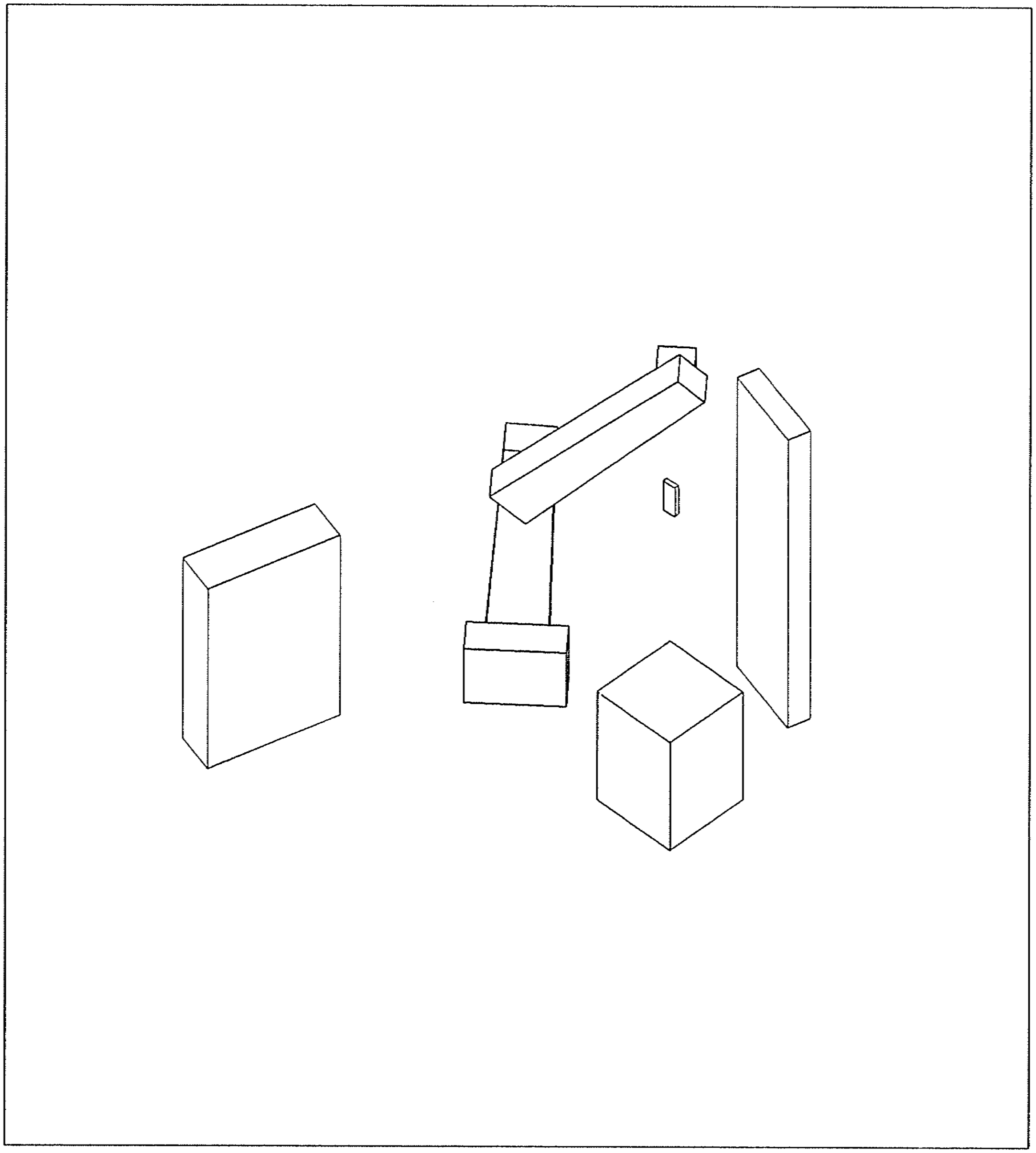
PATH 1 STEP 17
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



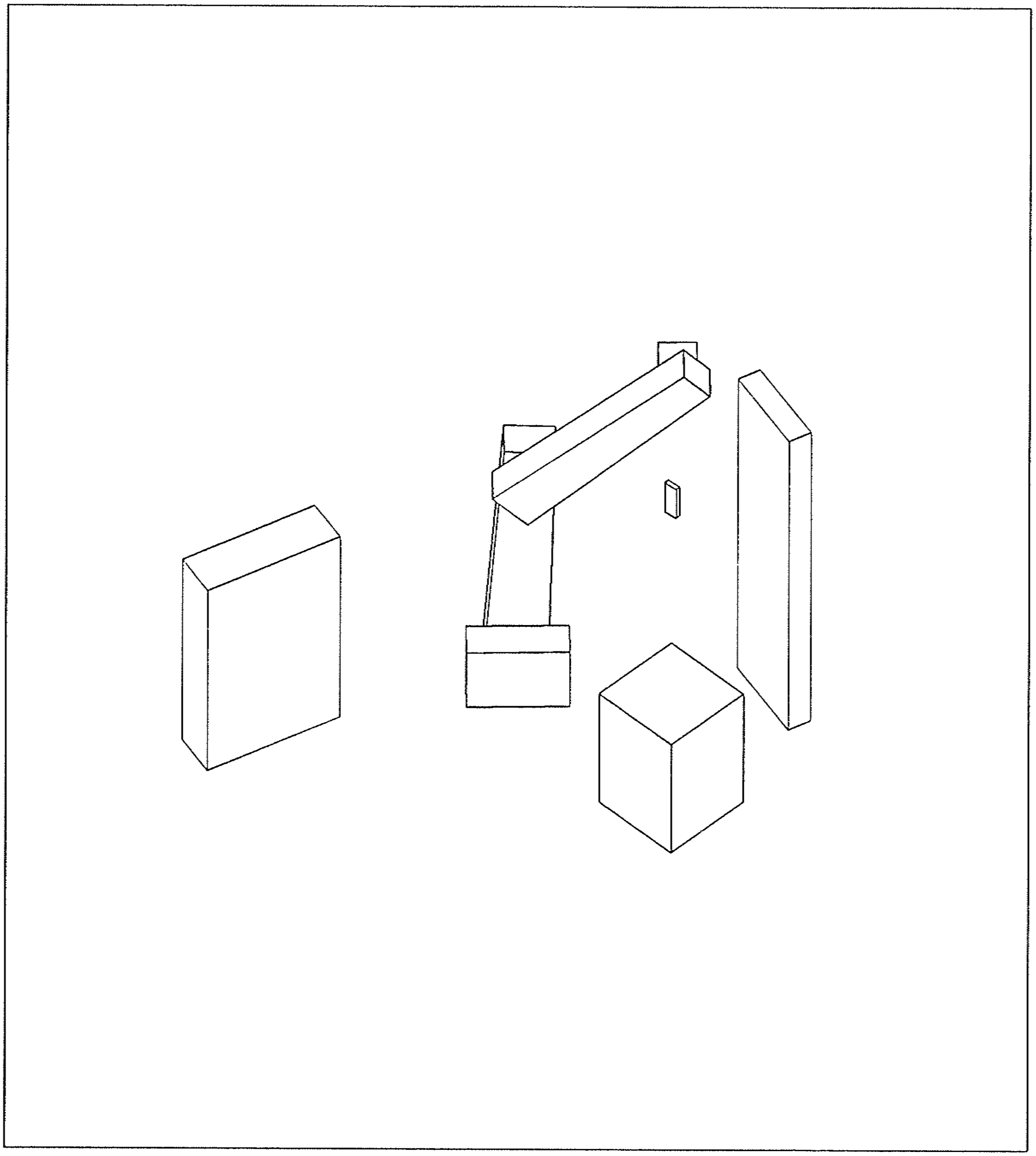
PATH 1 STEP 18
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



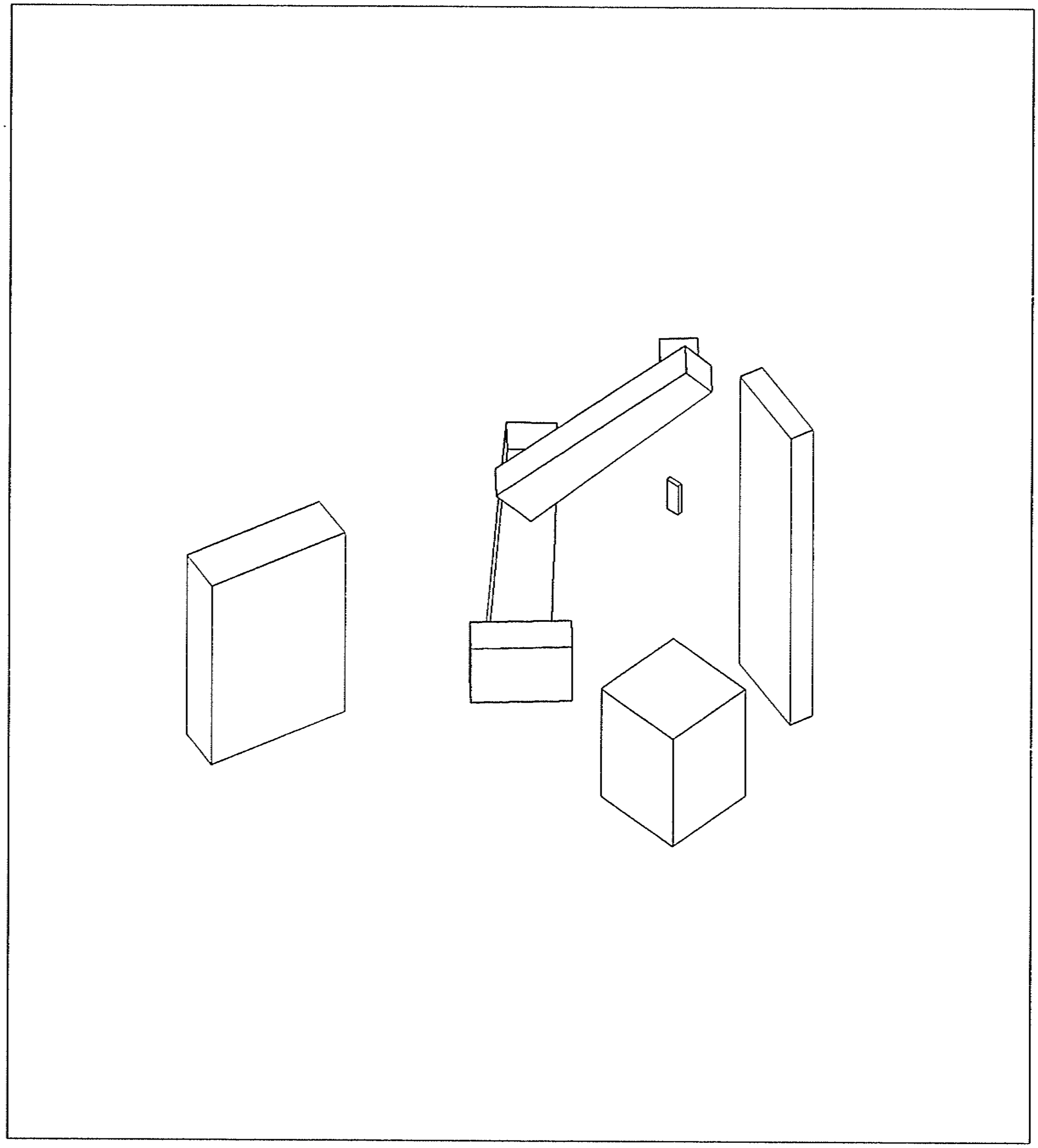
PATH 1 STEP 19
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



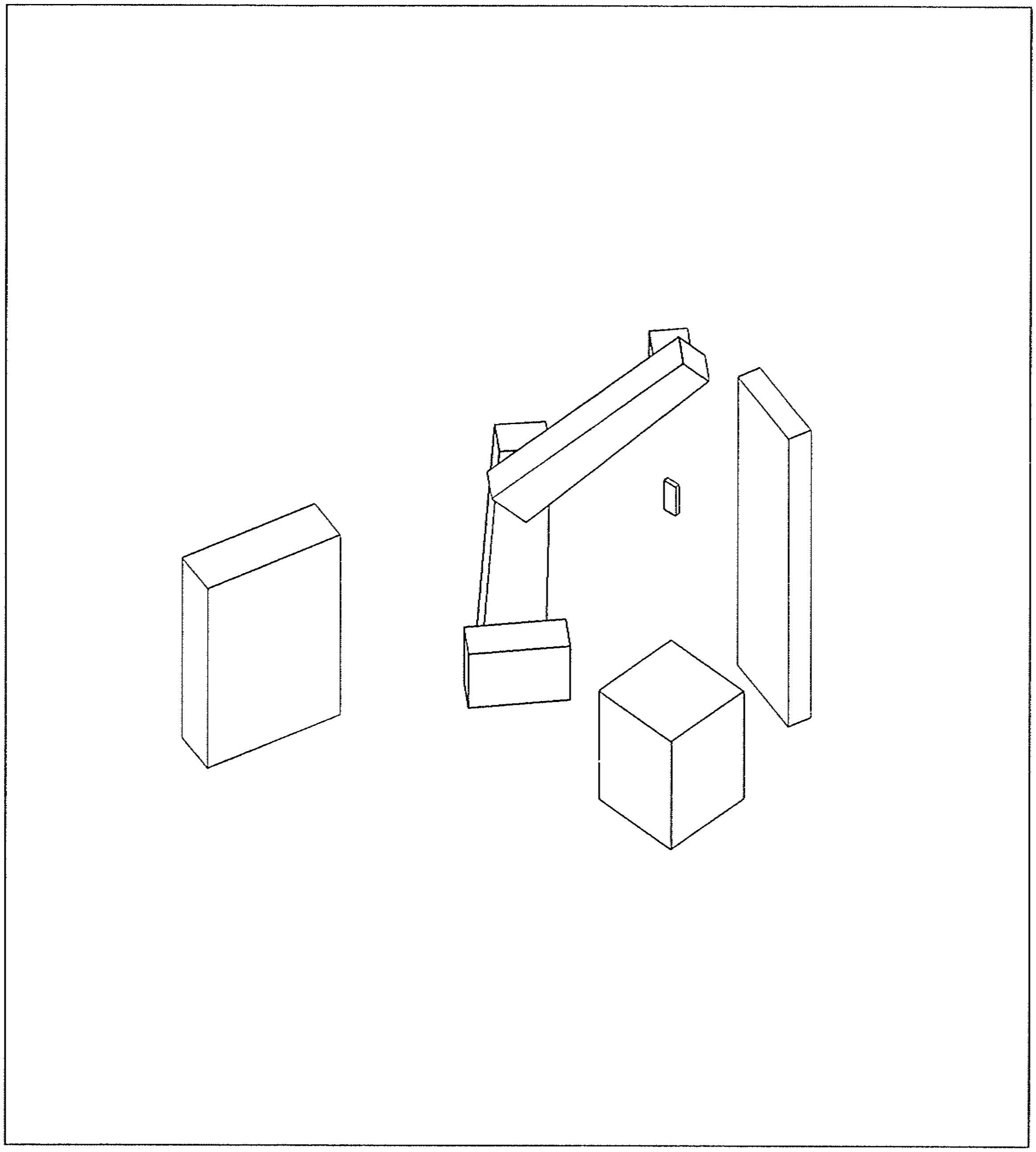
PATH 1 STEP 20
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



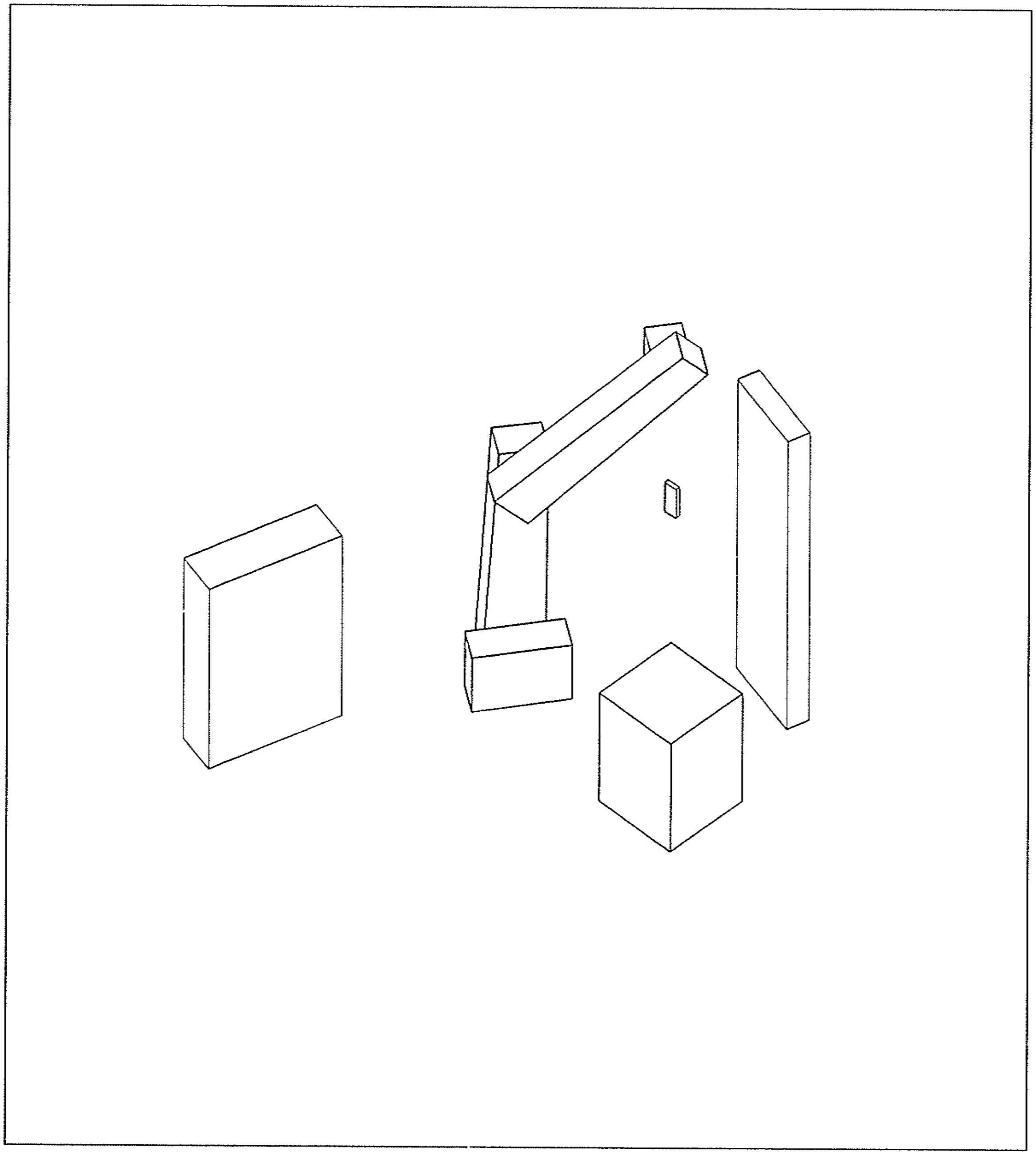
PATH 1 STEP 21
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



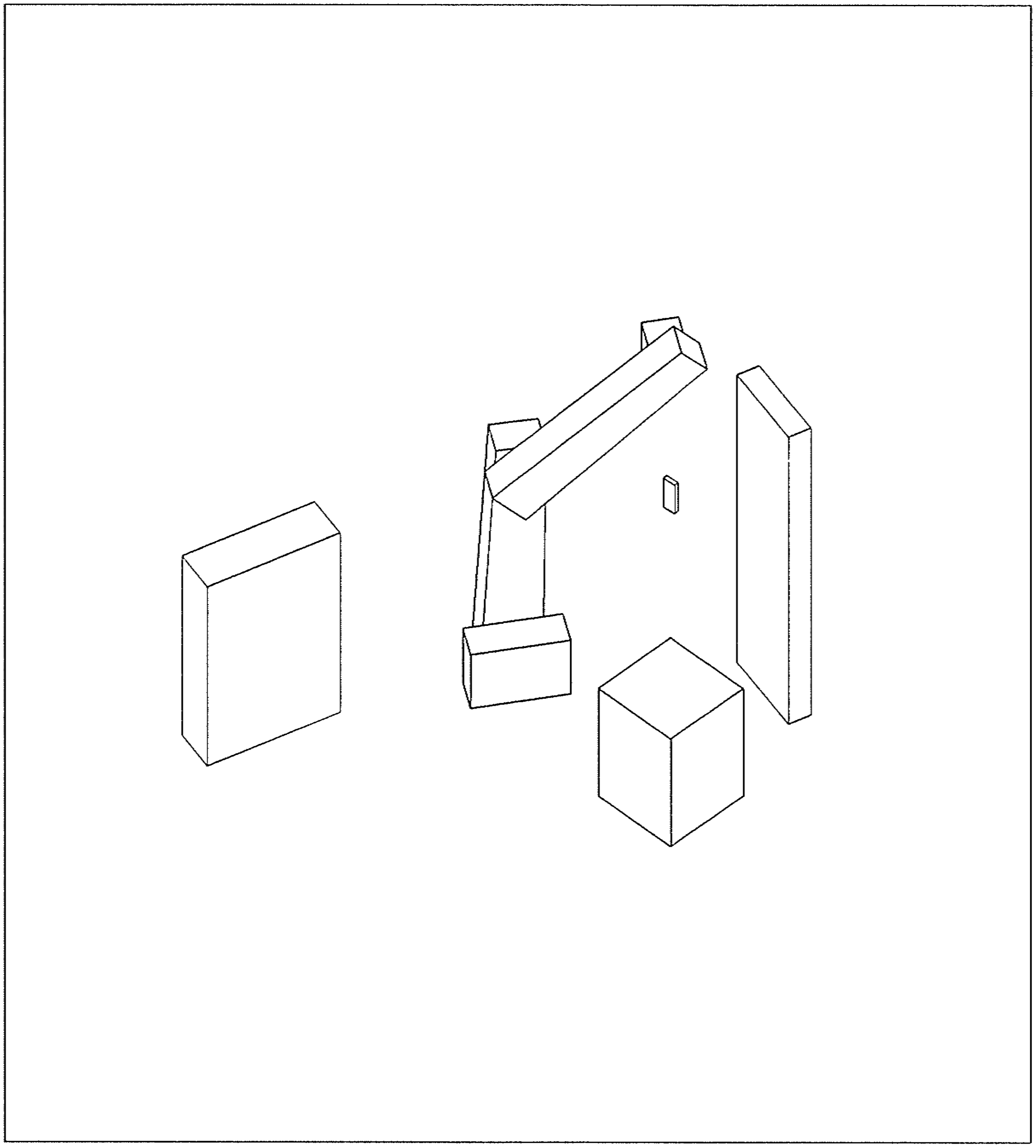
PATH 1 STEP 22
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



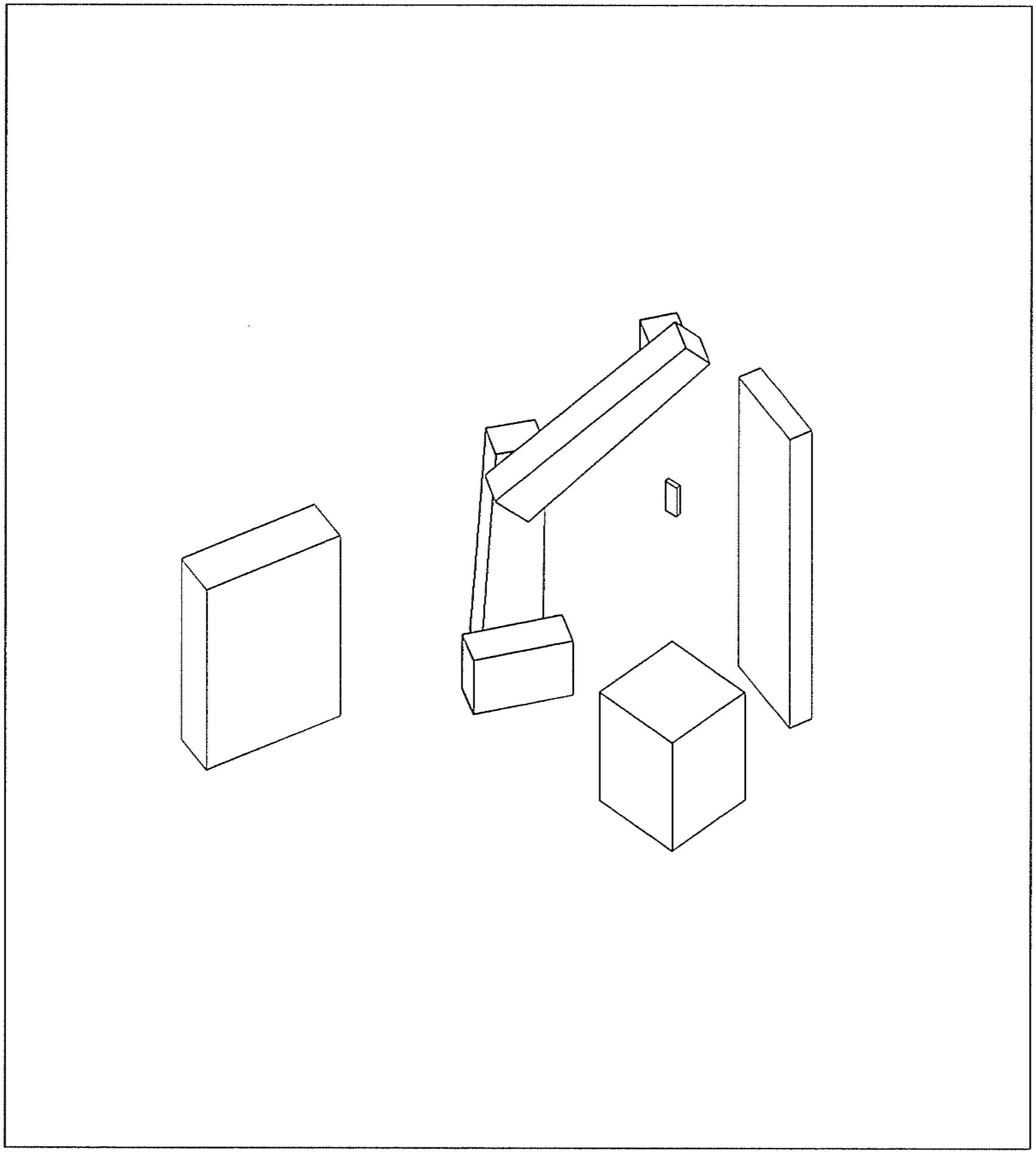
PATH 1 STEP 23
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



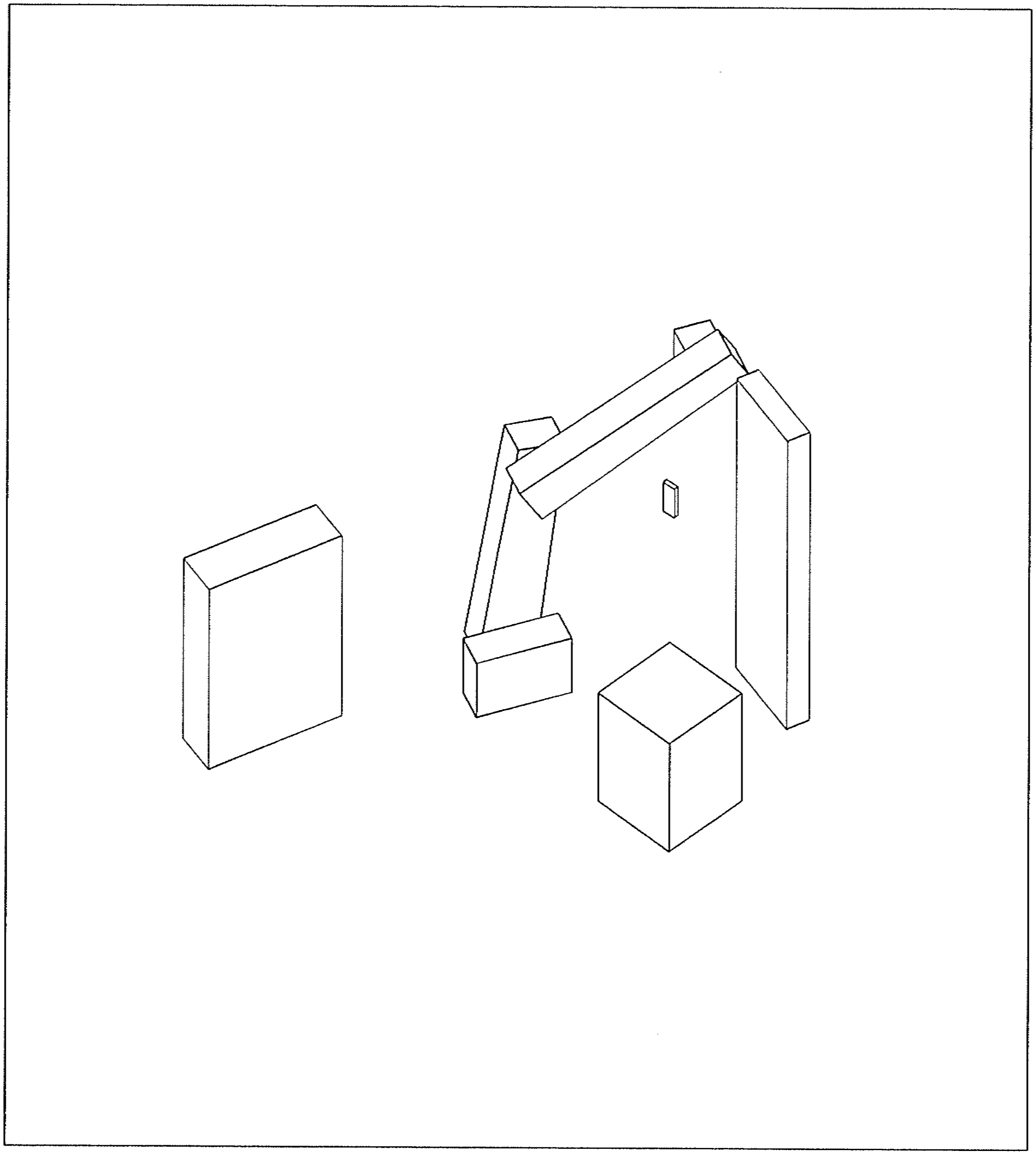
PATH 1 STEP 24
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



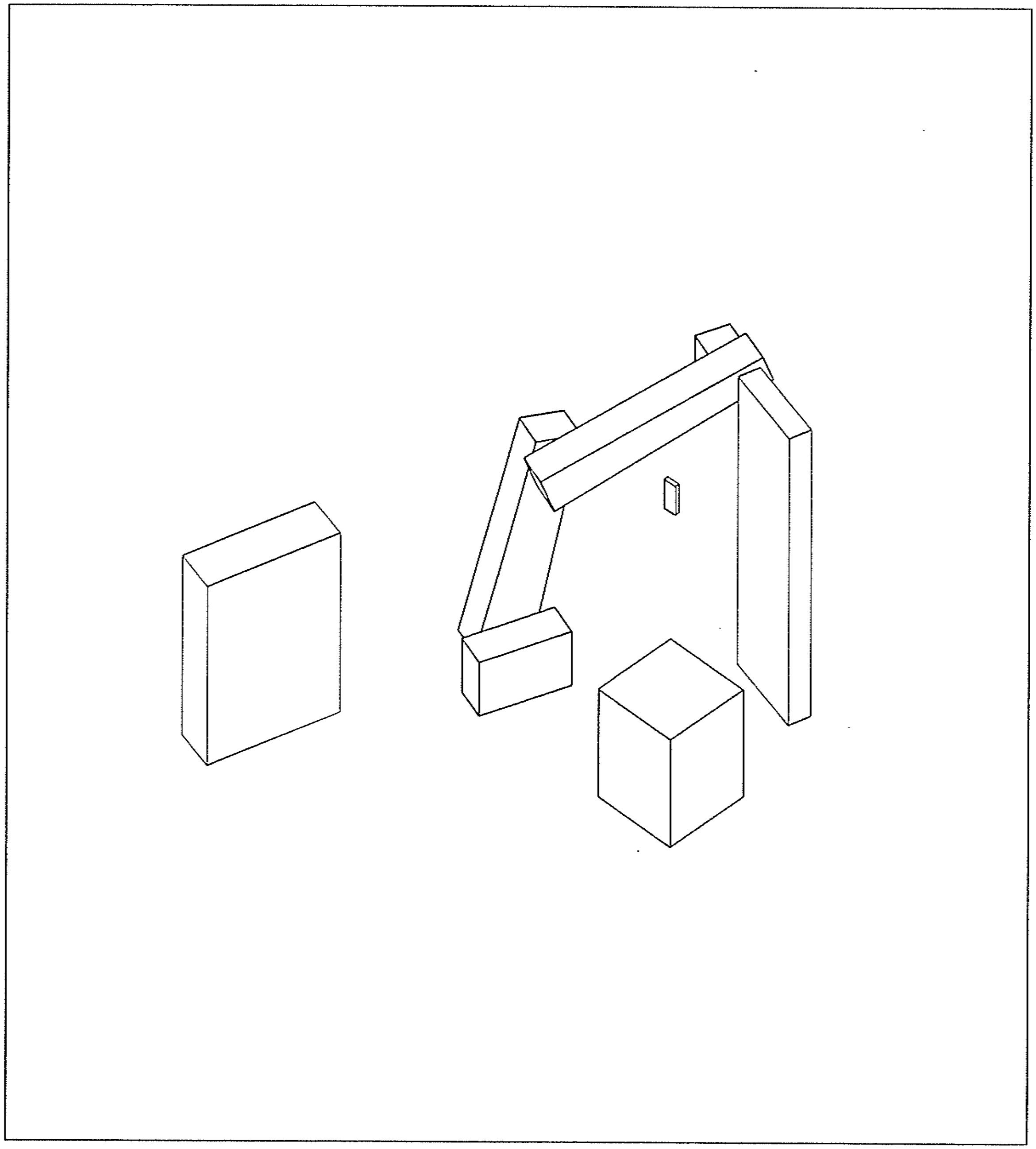
PATH 1 STEP 25
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



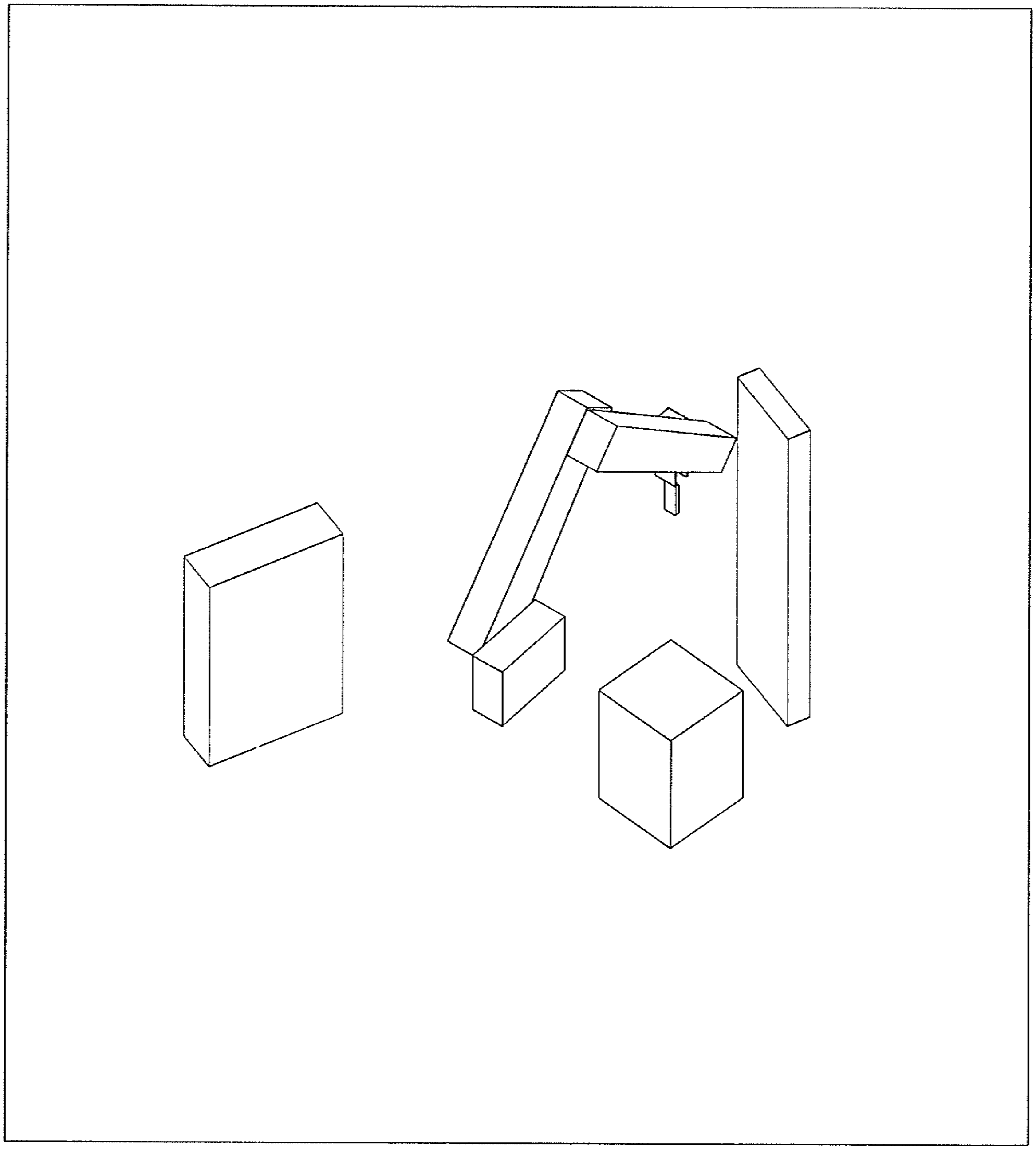
PATH 1 STEP 26
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



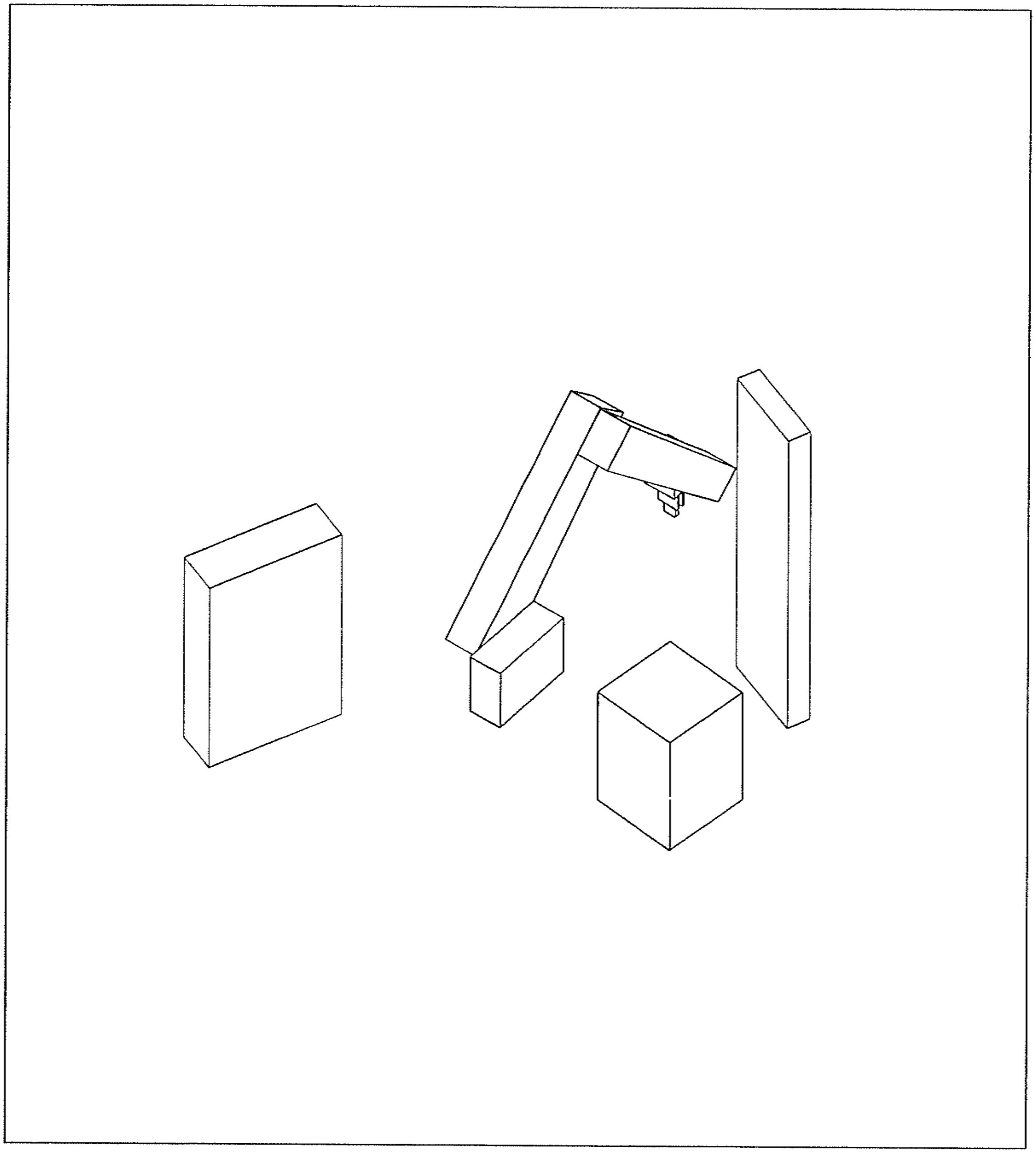
PATH 1 STEP 27
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



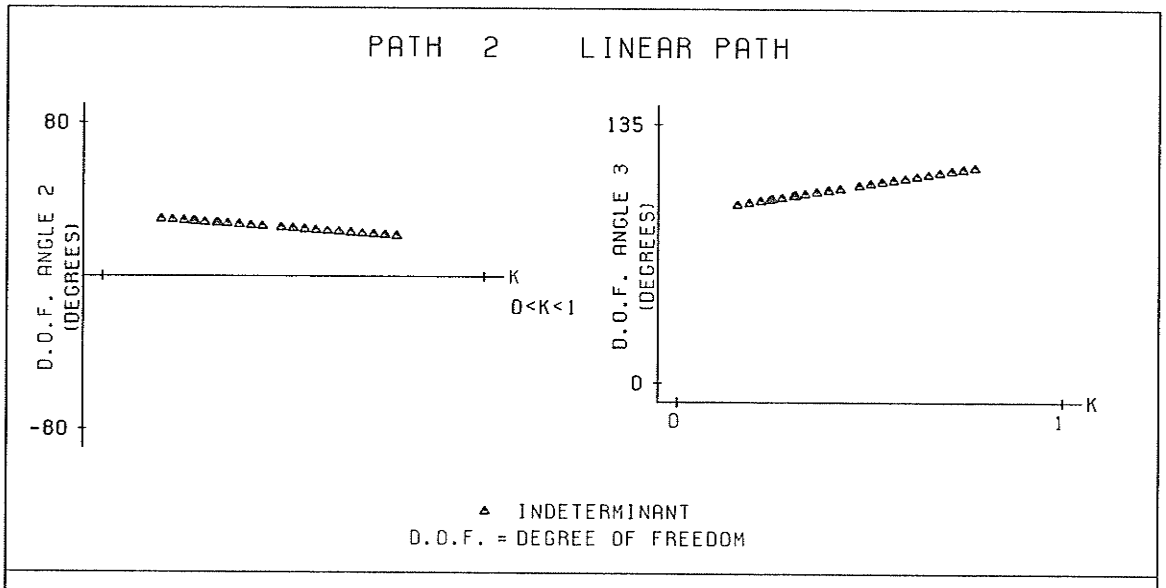
PATH 1 STEP 28
SAFE VARIABLE CONFIGURATION
PATH IS SAFE

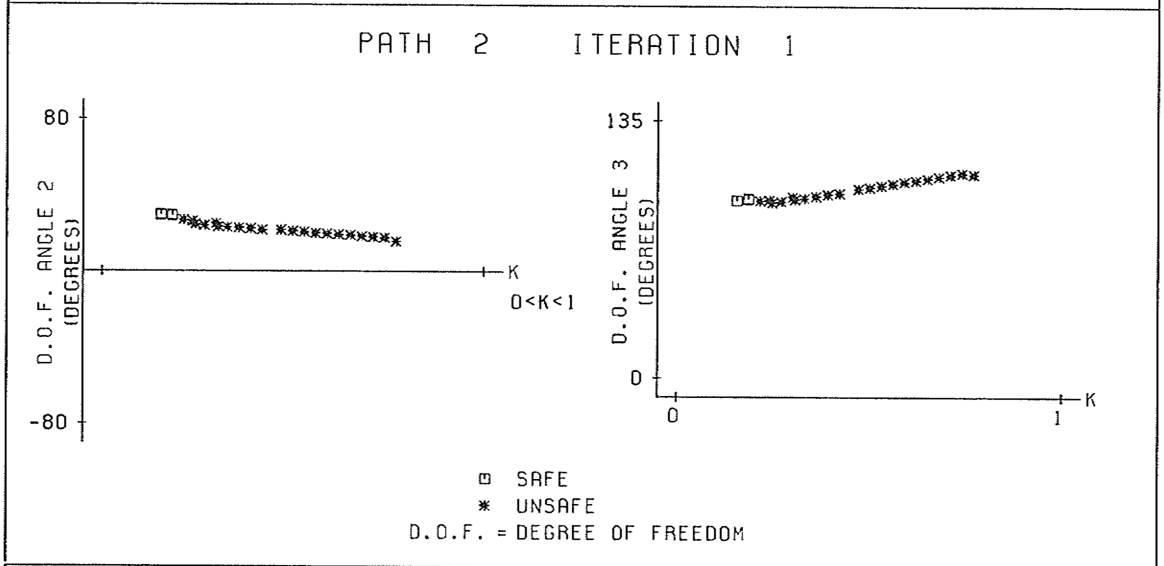


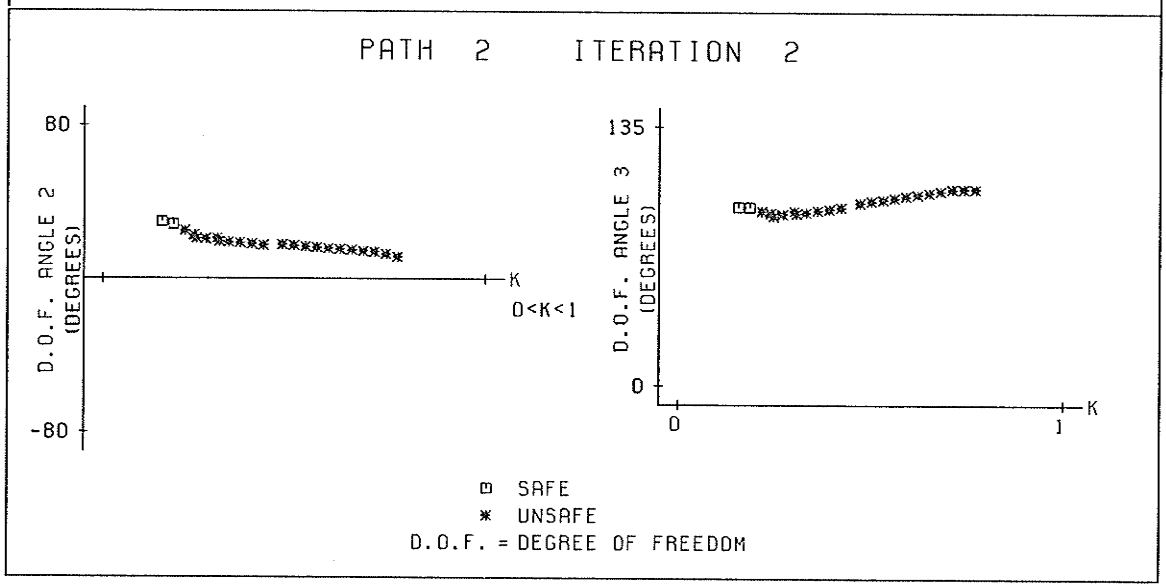
PATH 1 STEP 29
SAFE FIXED CONFIGURATION
PATH IS SAFE

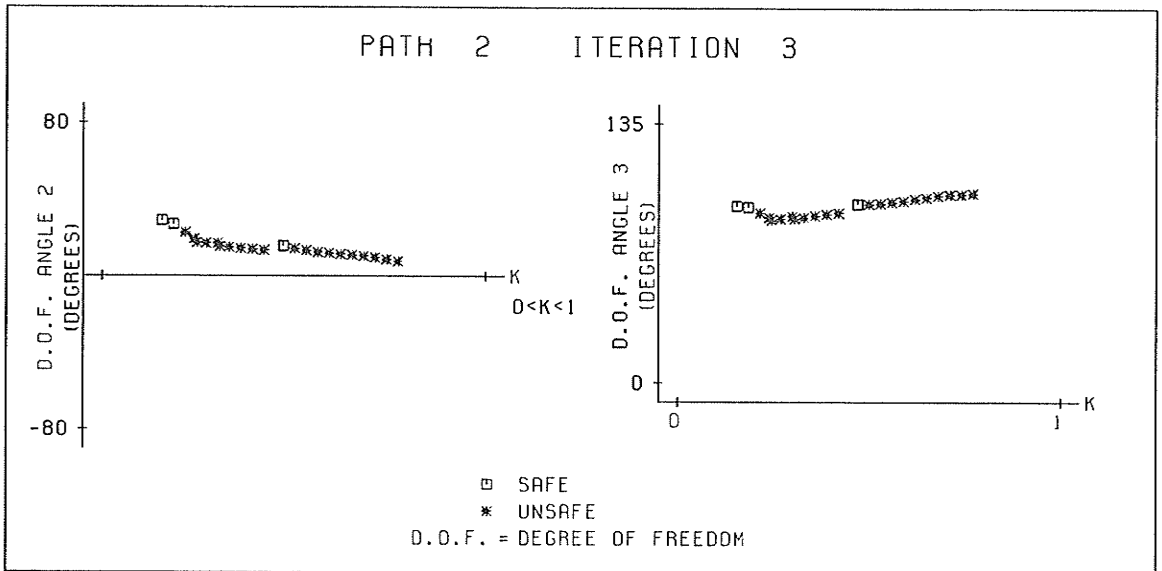


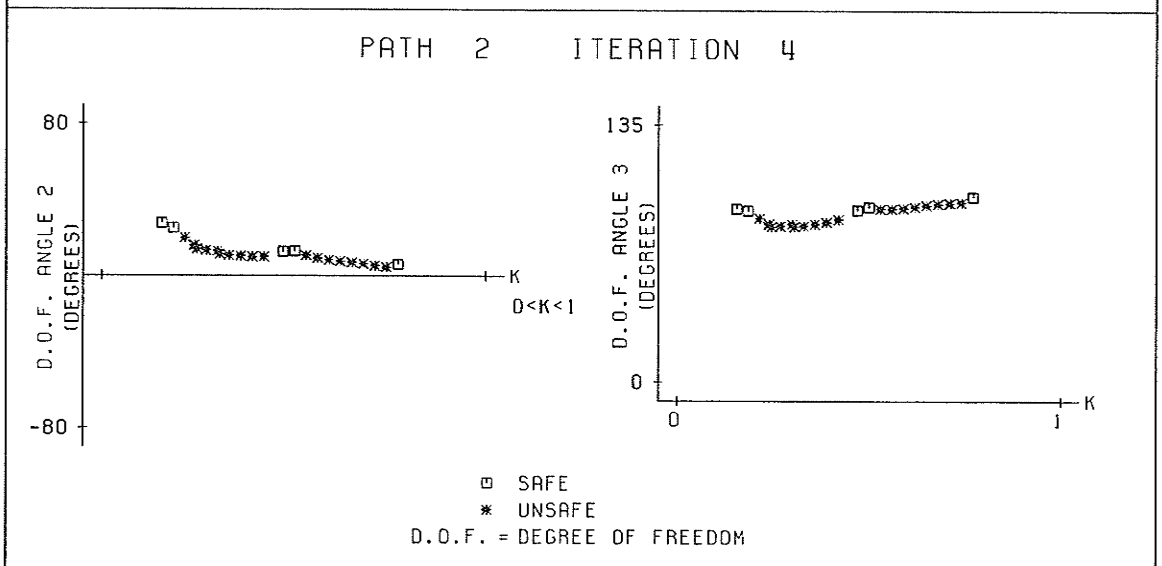
PATH 1 STEP 30
SAFE FIXED CONFIGURATION
PATH IS SAFE

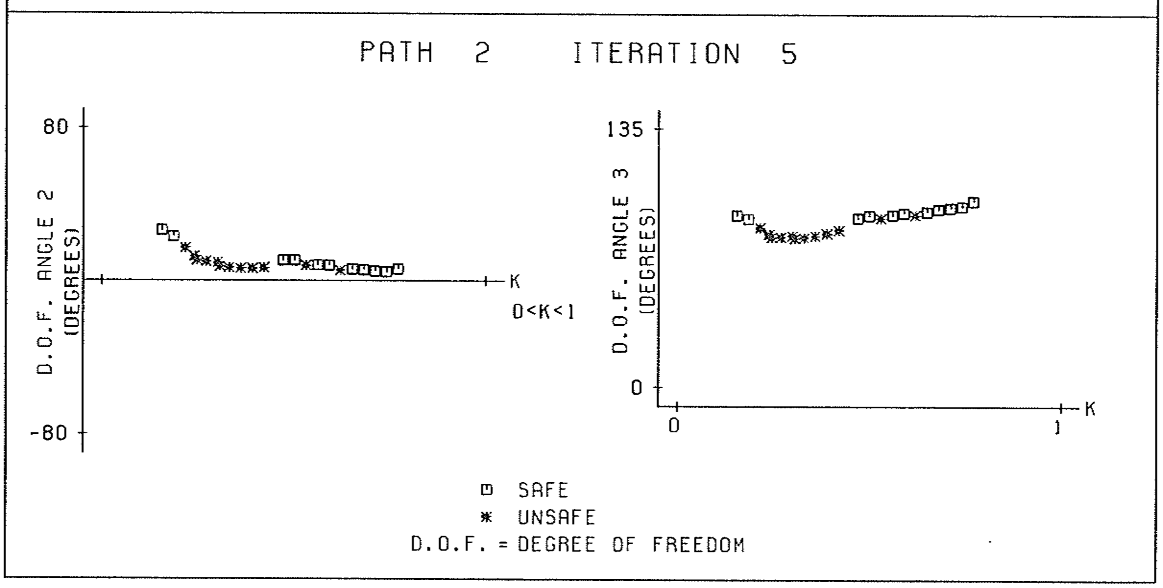

PATH 2 LINEAR PATH
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
INDETERMINANT
D.O.F. = DEGREE OF FREEDOM

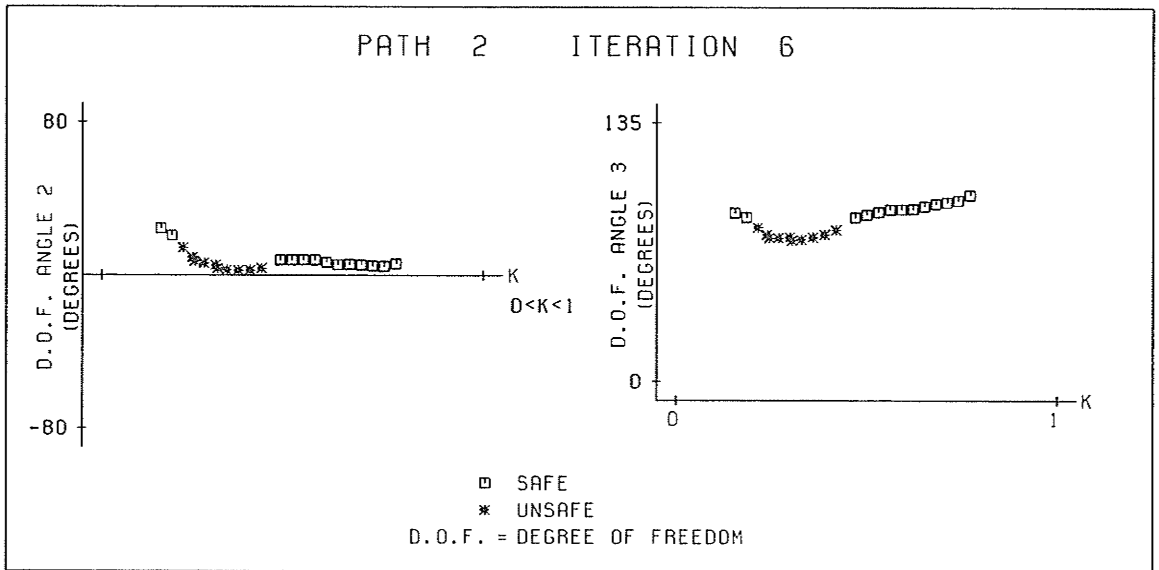

PATH 2 ITERATION 1
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

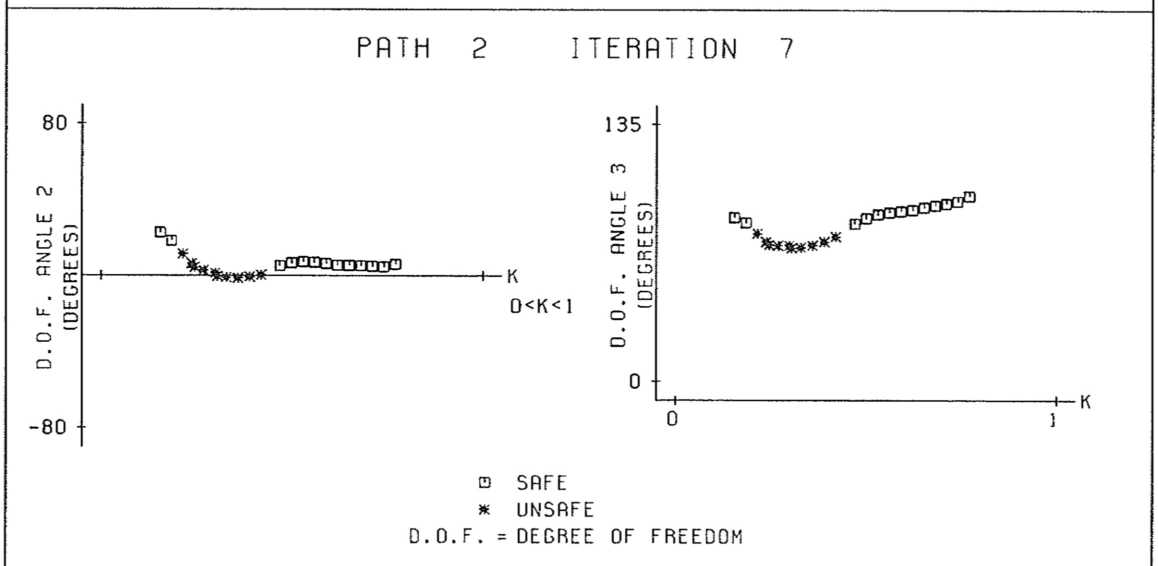

PATH 2 ITERATION 2
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

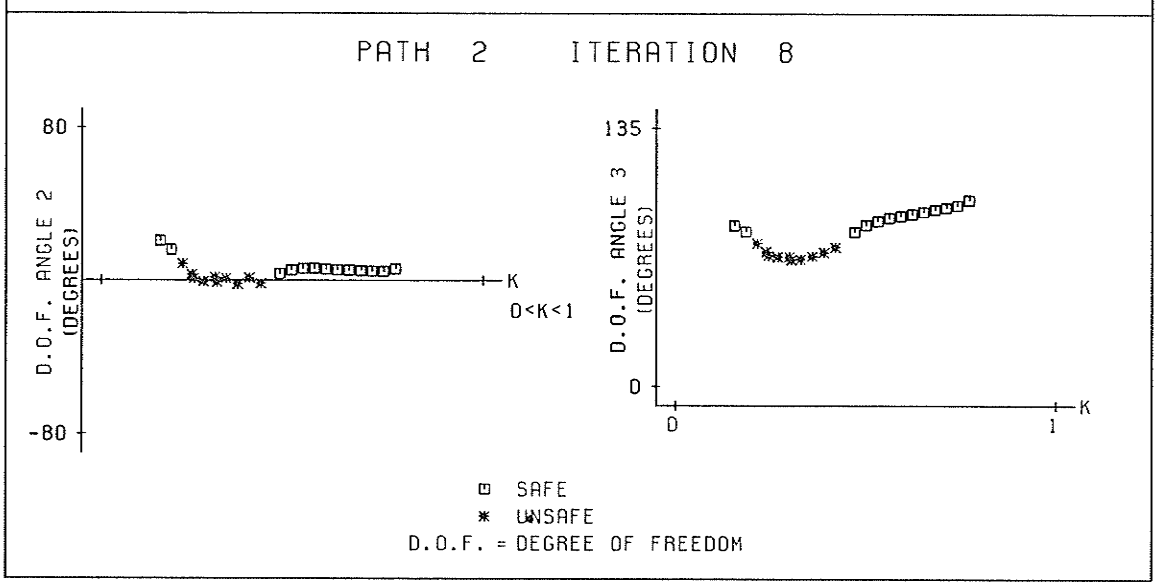

PATH 2 ITERATION 3
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM


PATH 2 ITERATION 4
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

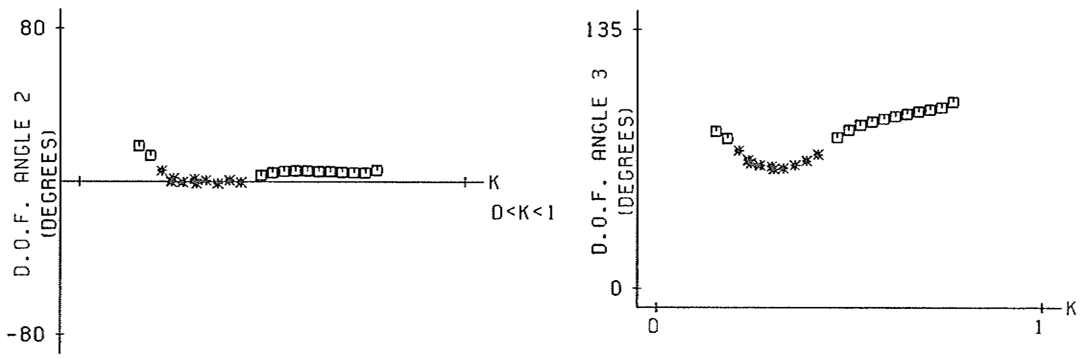

PATH 2 ITERATION 5
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM


PATH 2 ITERATION 6
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM


PATH 2 ITERATION 7
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

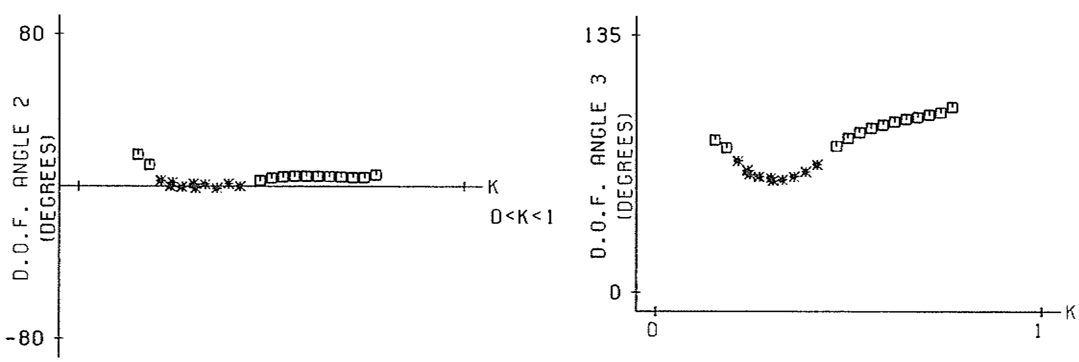

PATH 2 ITERATION 8
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

PATH 2 ITERATION 9



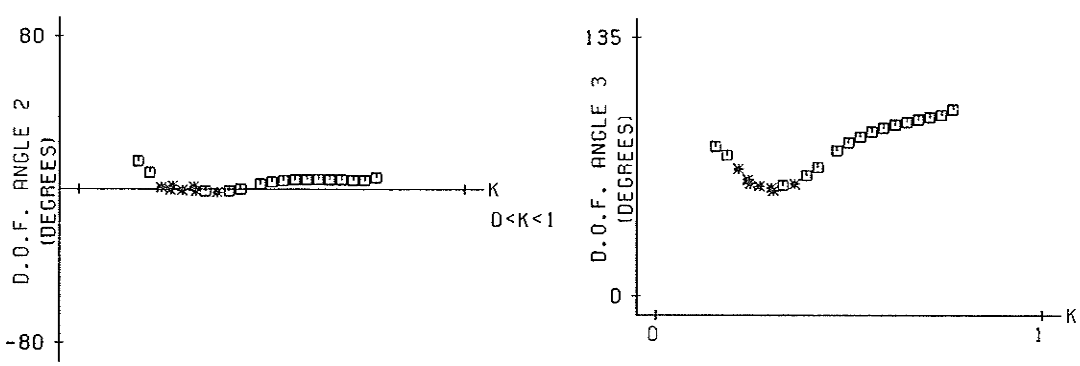
▫ SAFE
* UNSAFE
D.O.F. = DEGREE OF FREEDOM

PATH 2 ITERATION 10



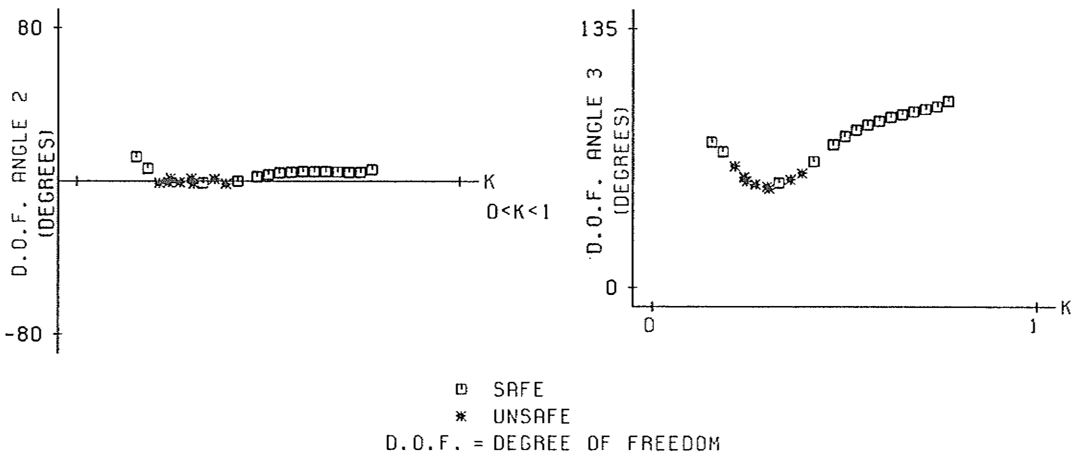
▫ SAFE
* UNSAFE
D.O.F. = DEGREE OF FREEDOM

PATH 2 ITERATION 11

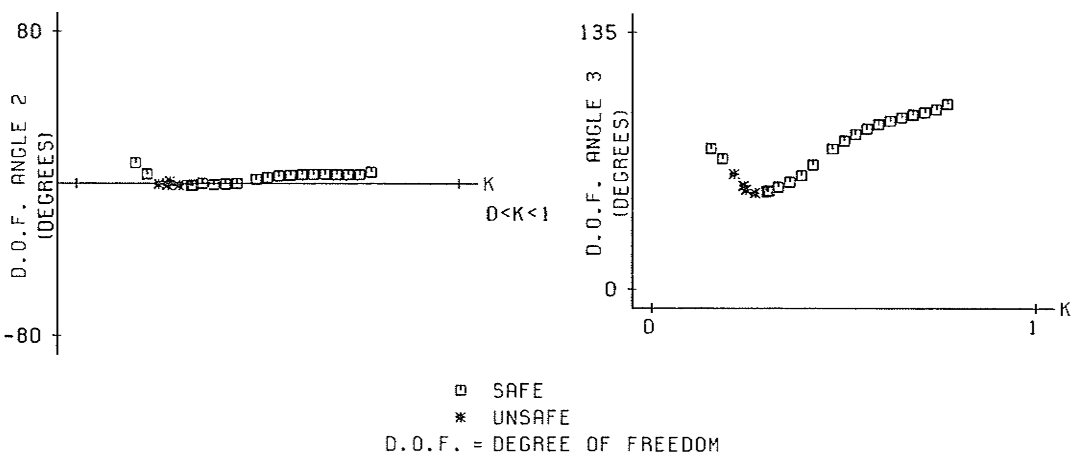


▫ SAFE
* UNSAFE
D.O.F. = DEGREE OF FREEDOM

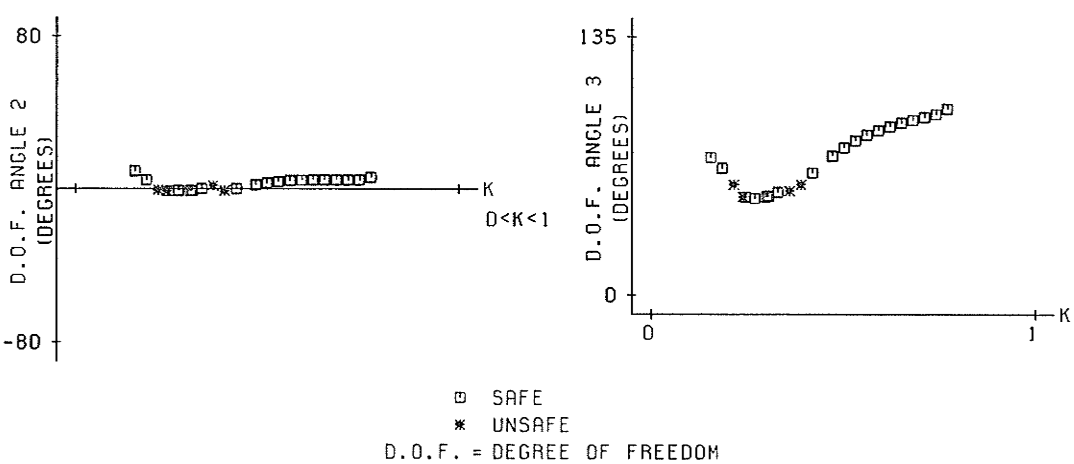
PATH 2 ITERATION 12



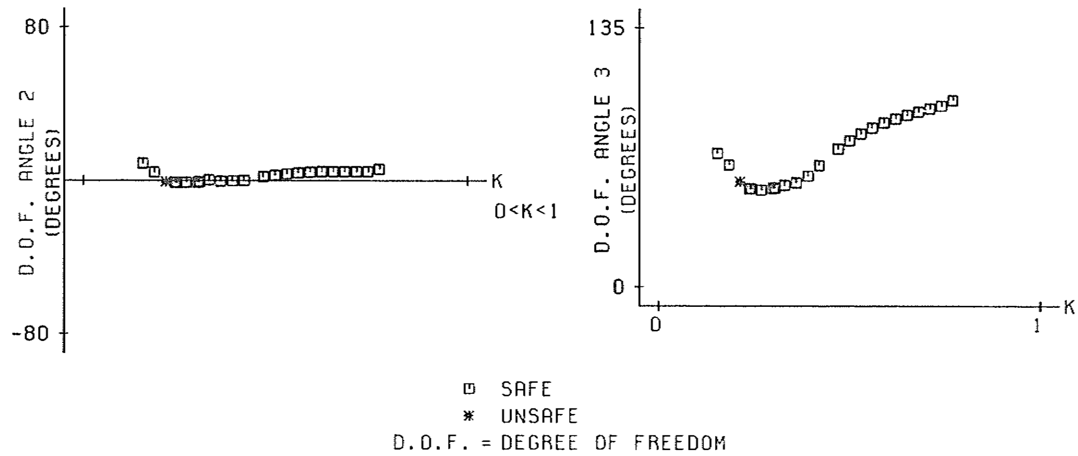
PATH 2 ITERATION 13



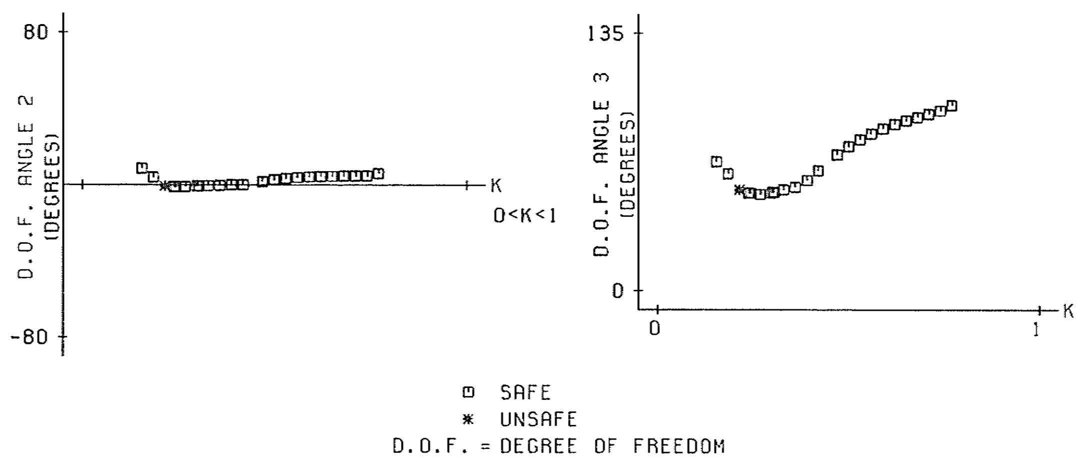
PATH 2 ITERATION 14



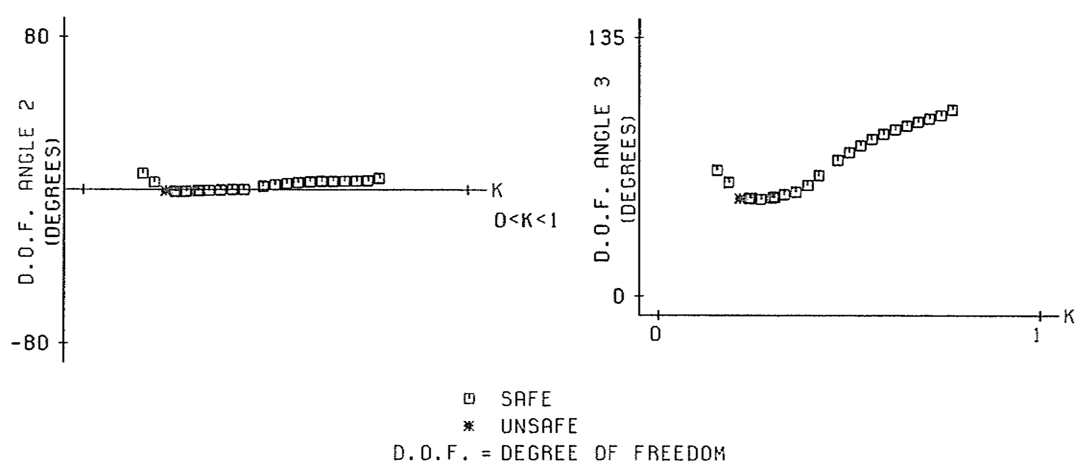
PATH 2 ITERATION 15

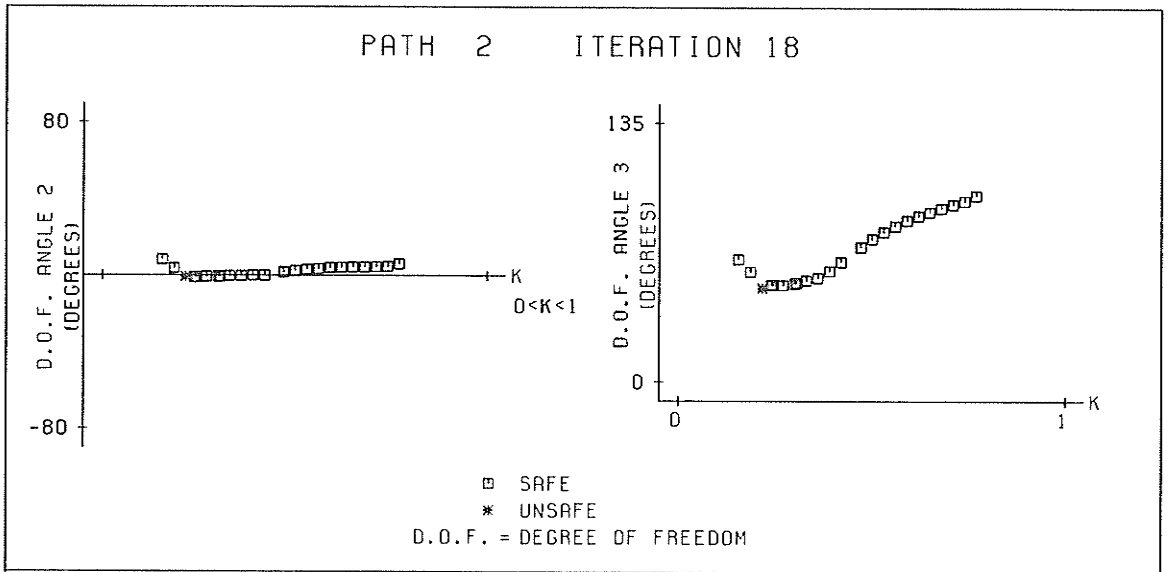


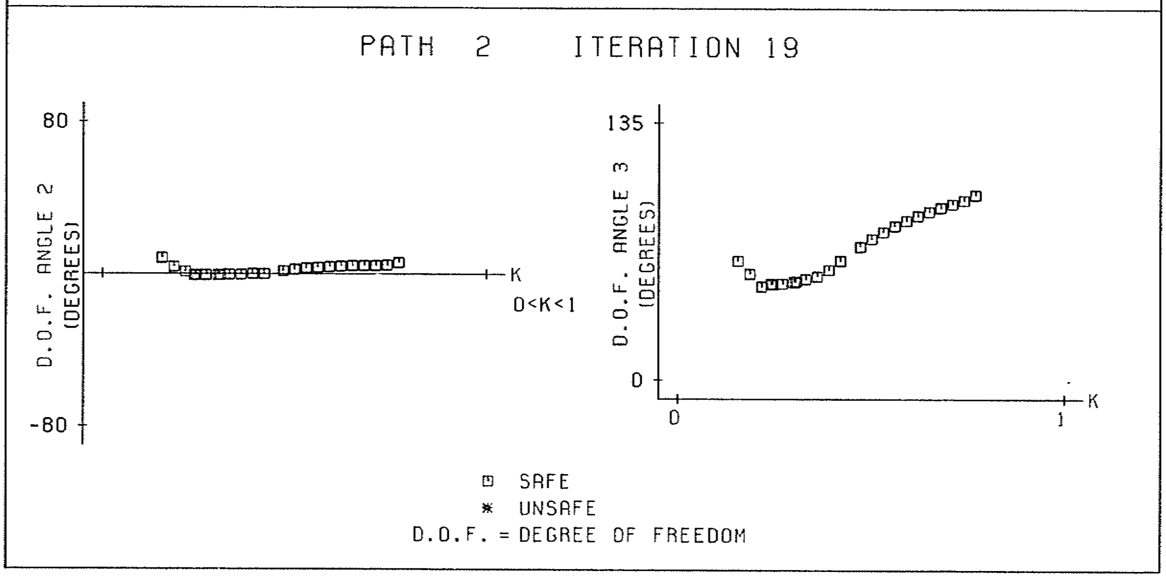
PATH 2 ITERATION 16

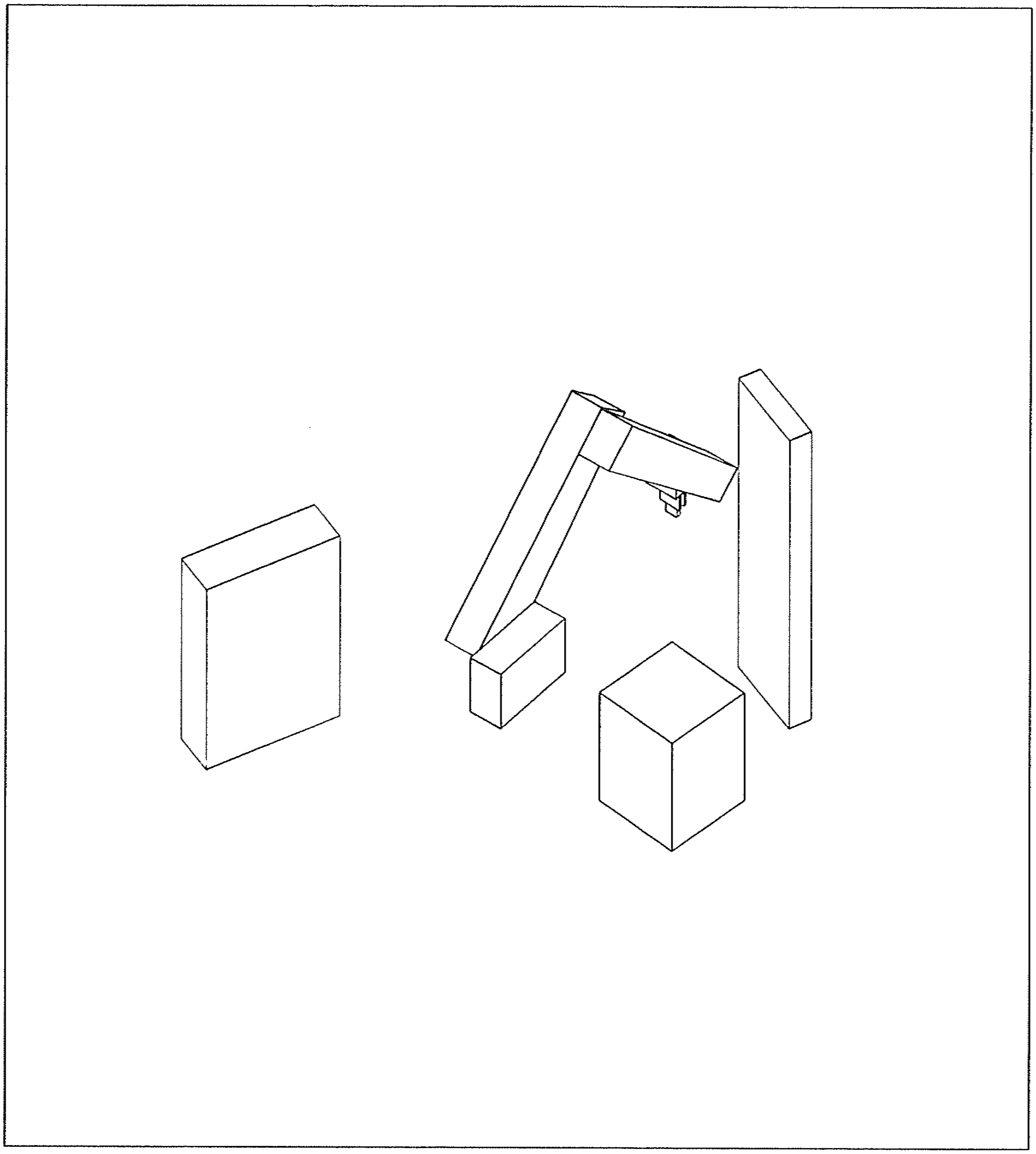


PATH 2 ITERATION 17

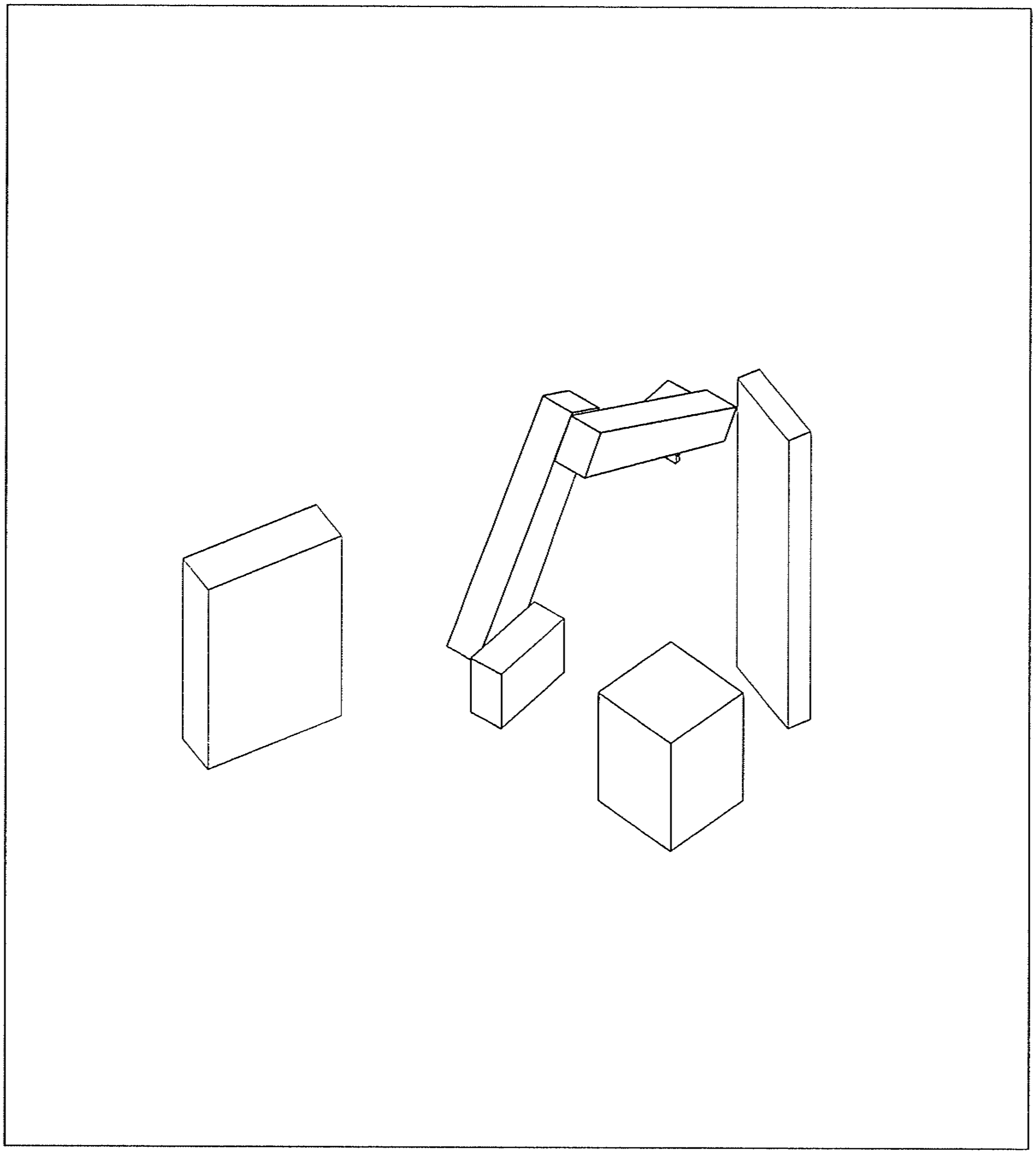



PATH 2 ITERATION 18
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

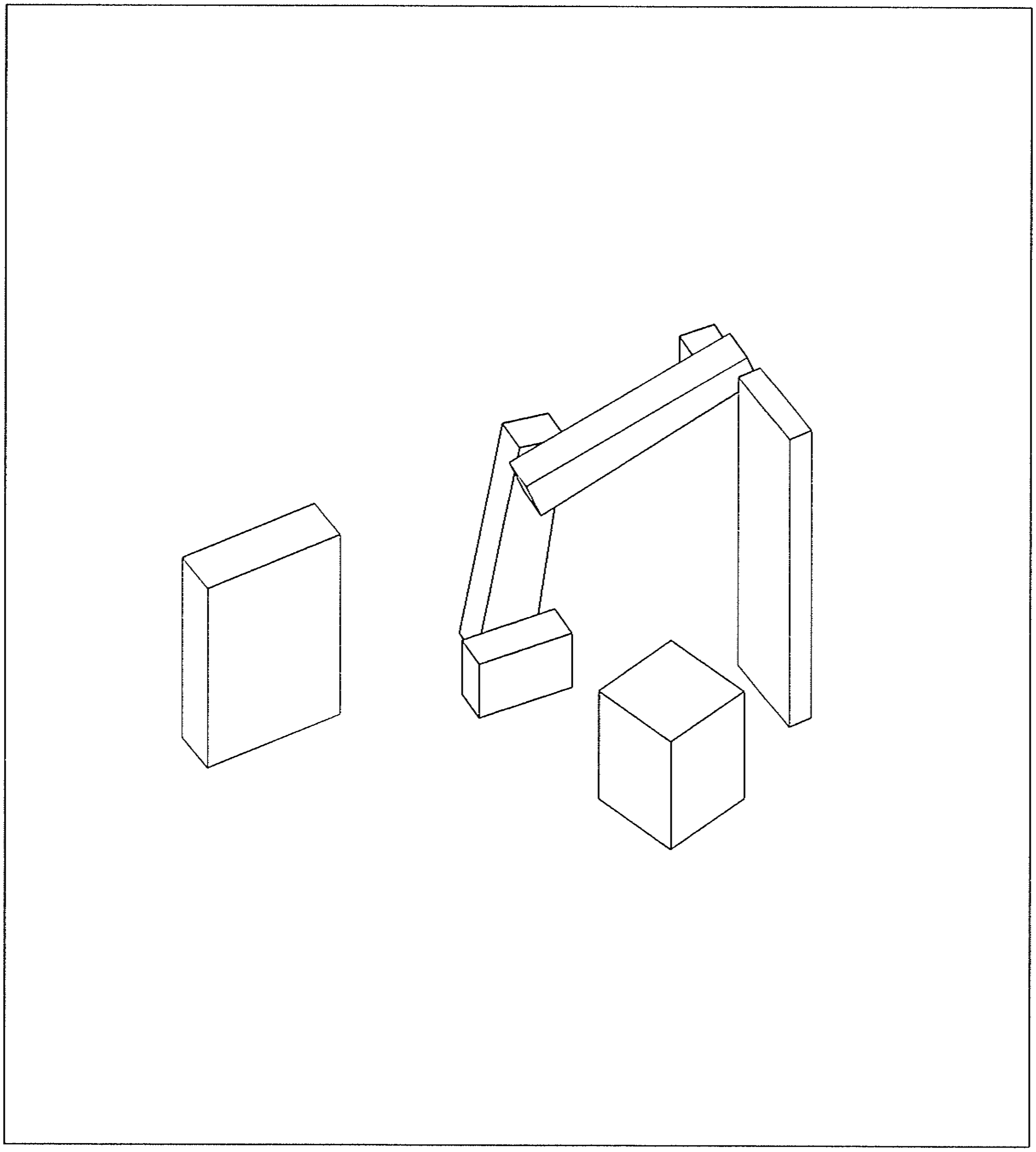

PATH 2 ITERATION 19
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
0
1
K
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM



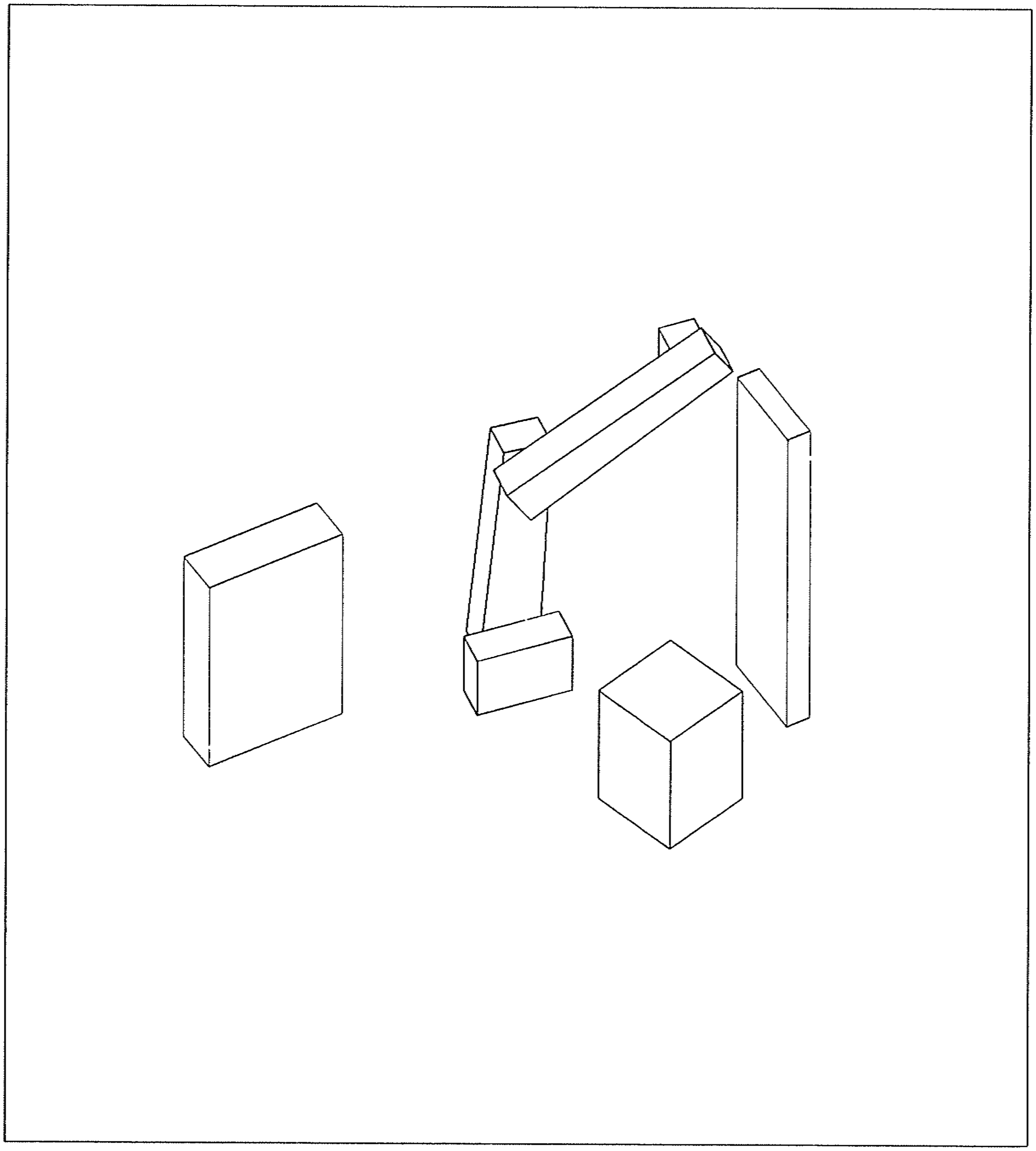
PATH 2 STEP 1
SAFE FIXED CONFIGURATION
PATH IS SAFE



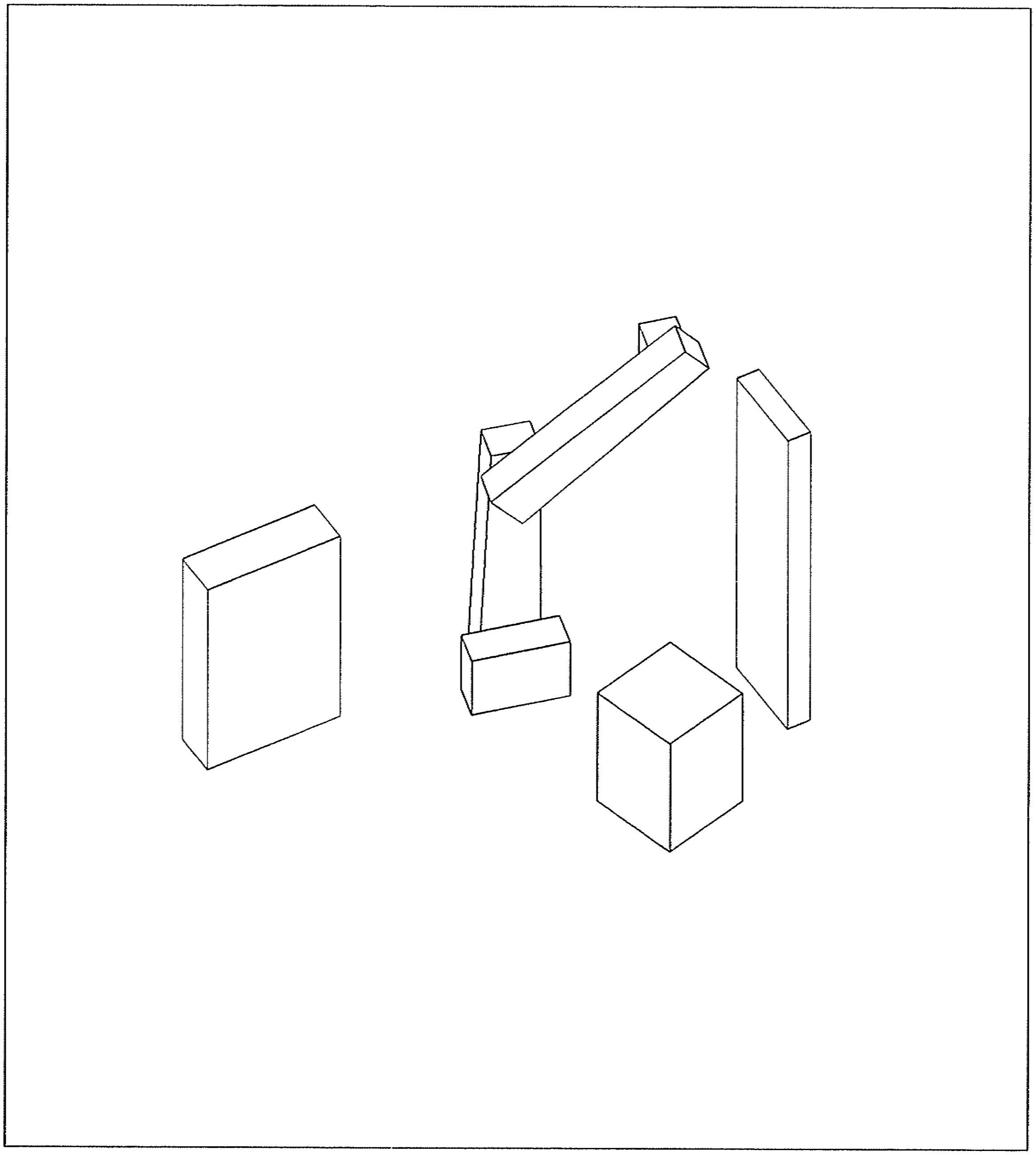
PATH 2 STEP 2
SAFE FIXED CONFIGURATION
PATH IS SAFE



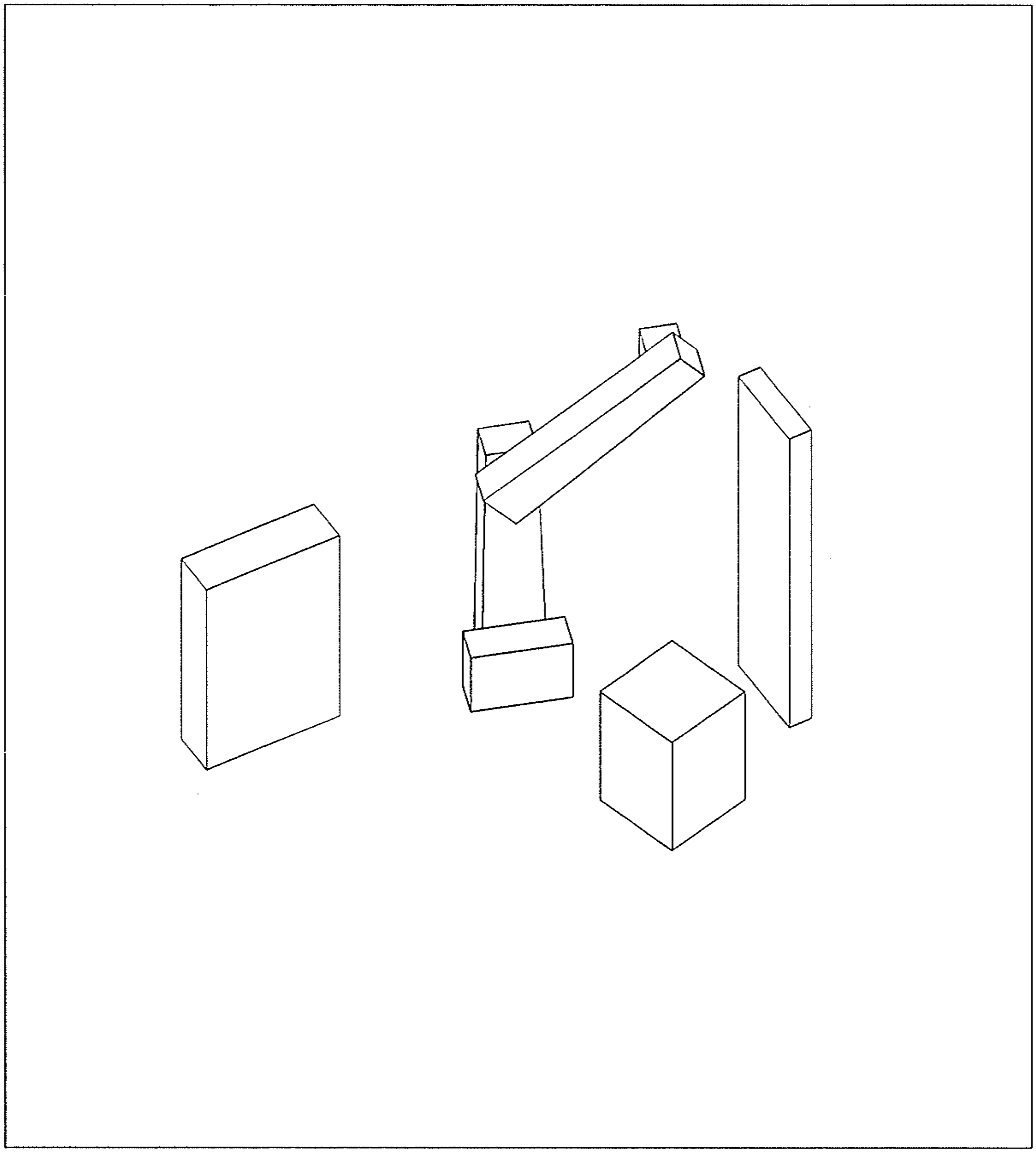
PATH 2 STEP 3
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



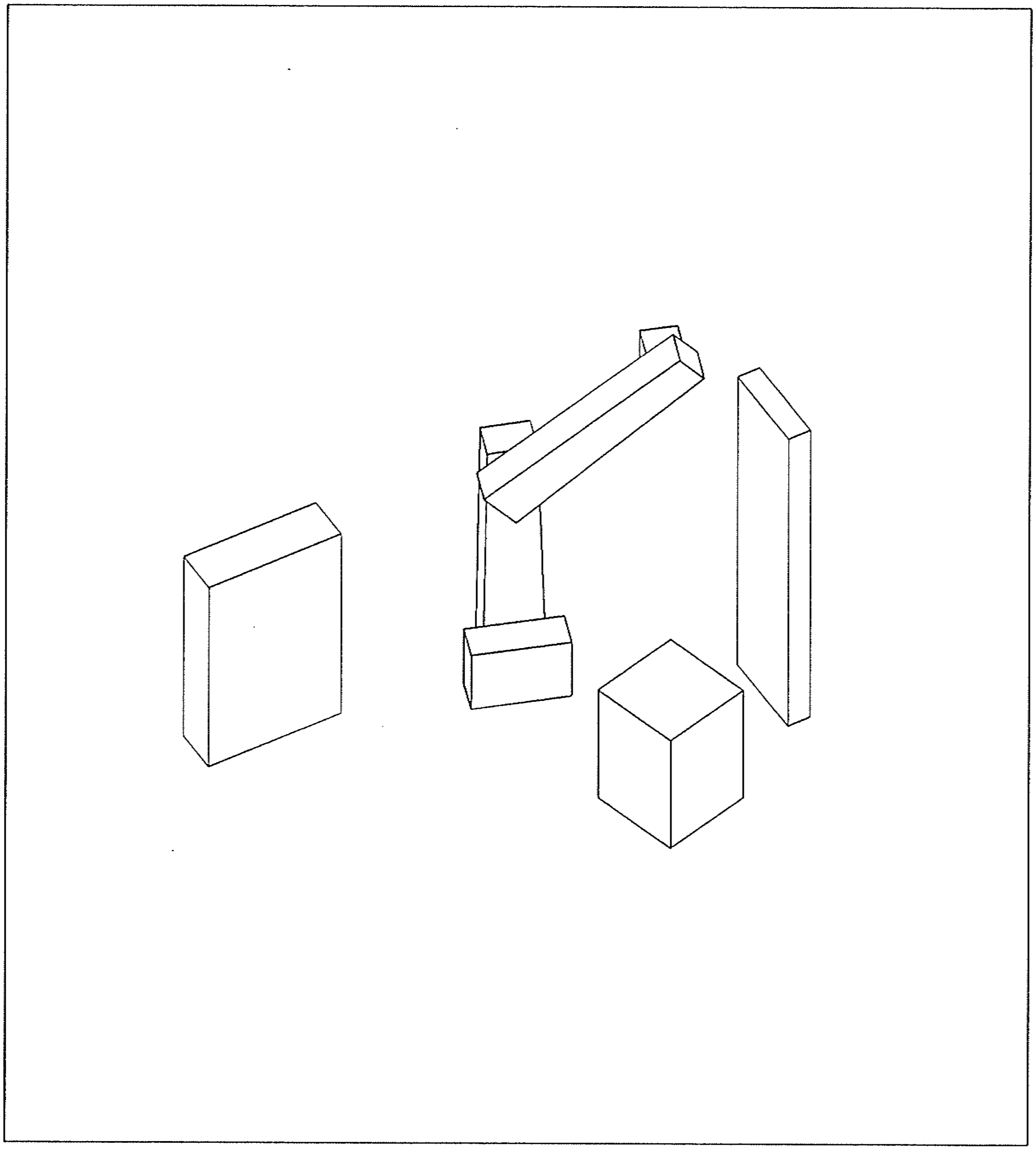
PATH 2 STEP 4
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



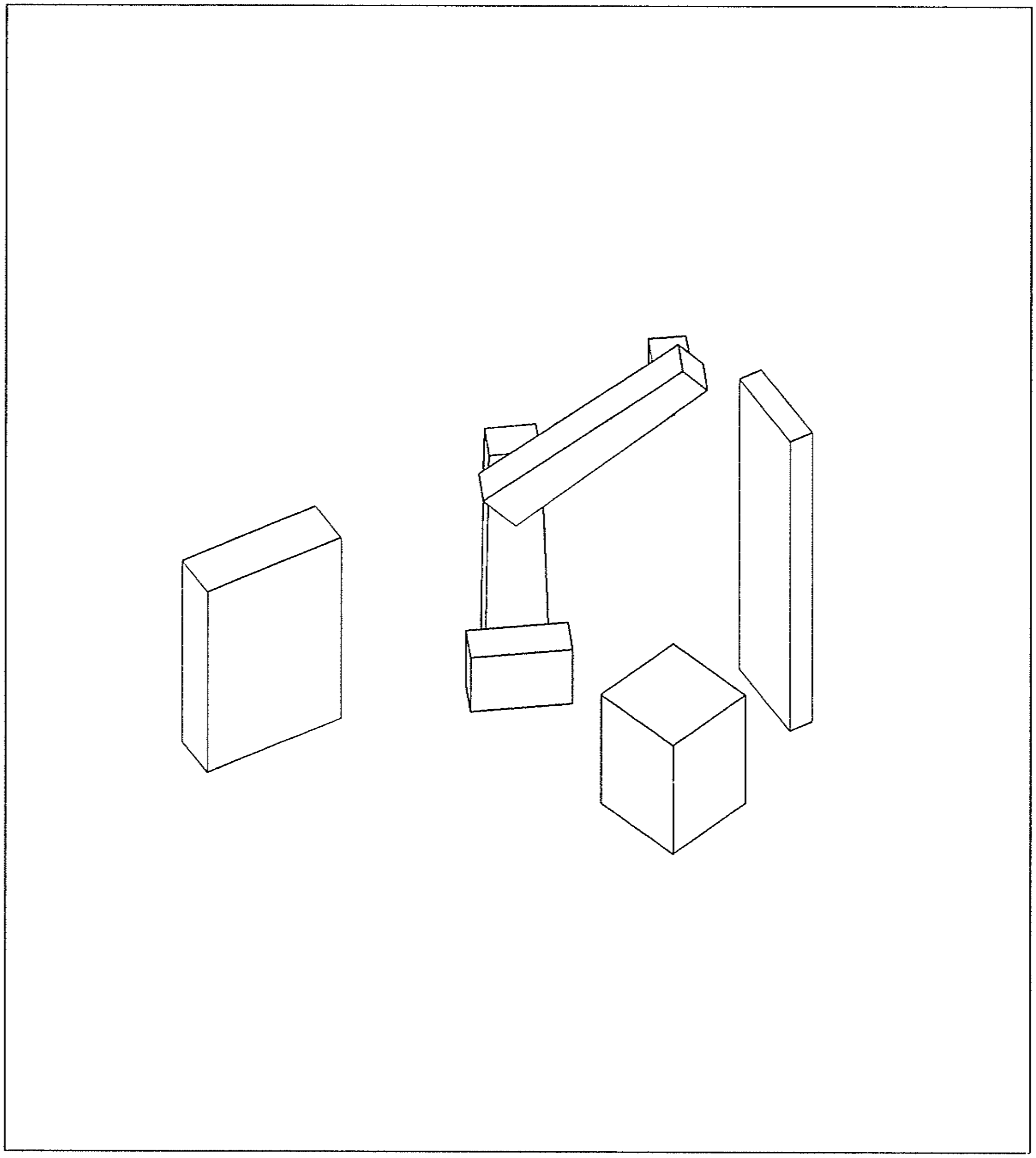
PATH 2 STEP 5
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



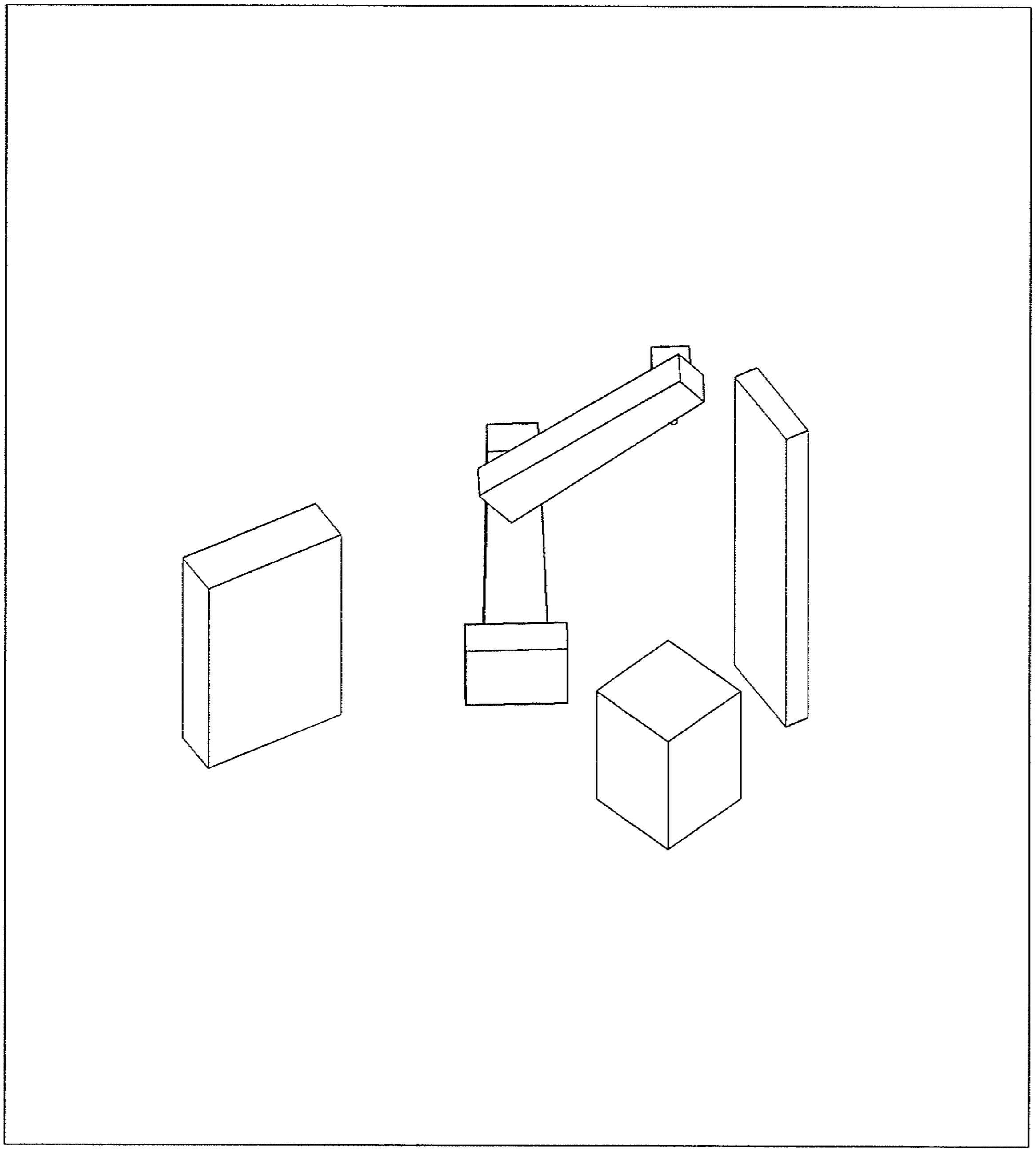
PATH 2 STEP 6
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



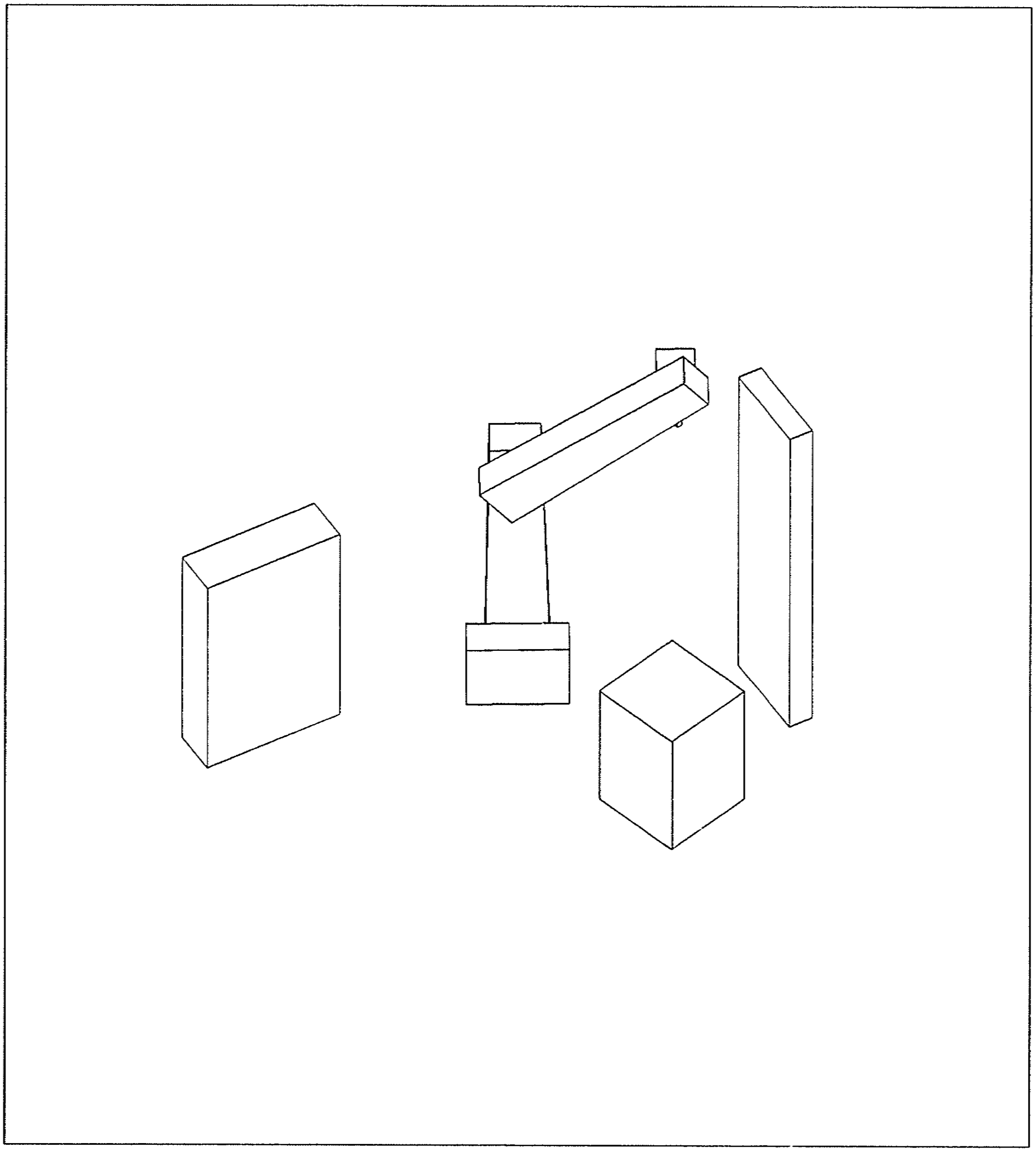
PATH 2 STEP 7
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



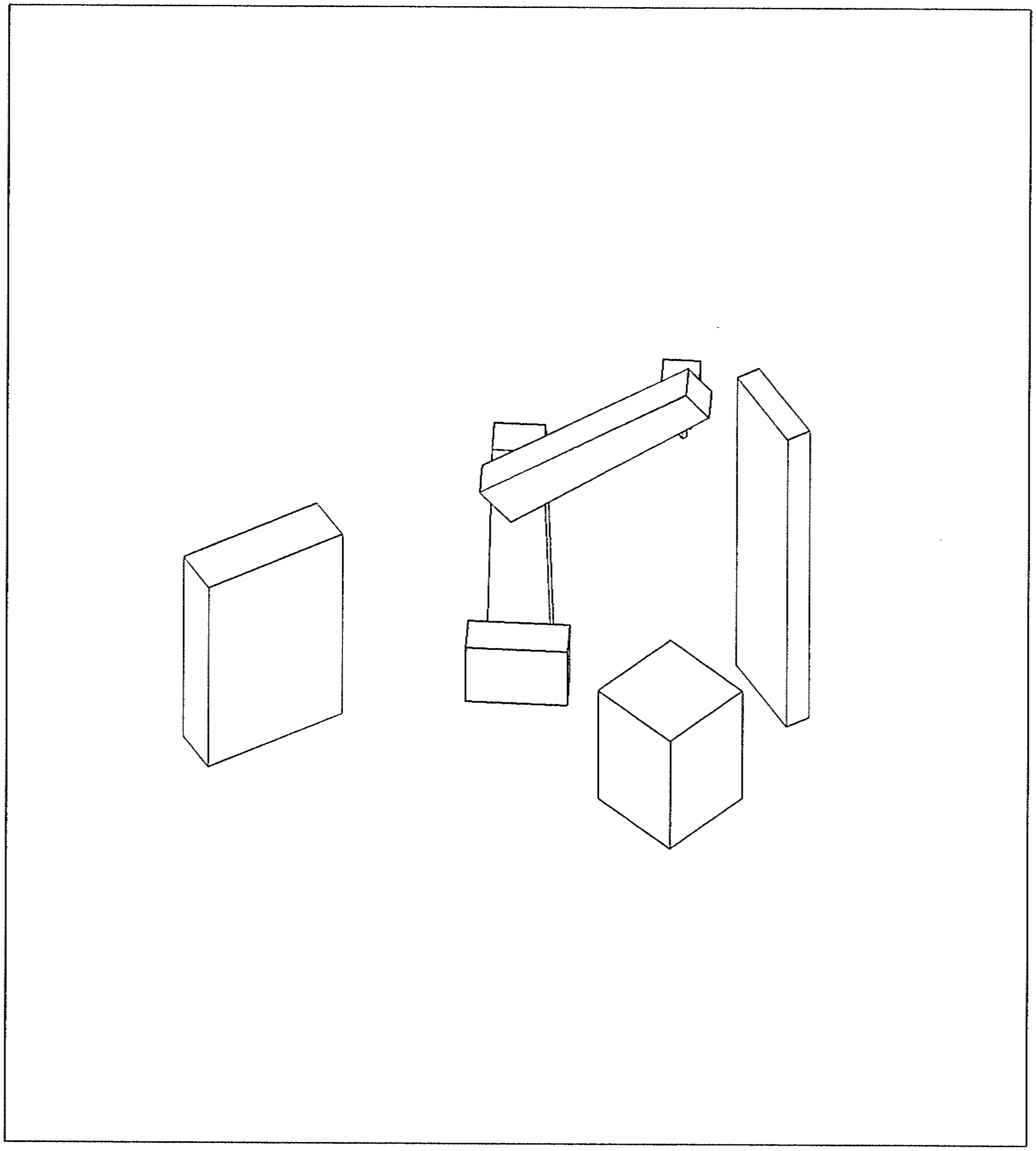
PATH 2 STEP 8
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



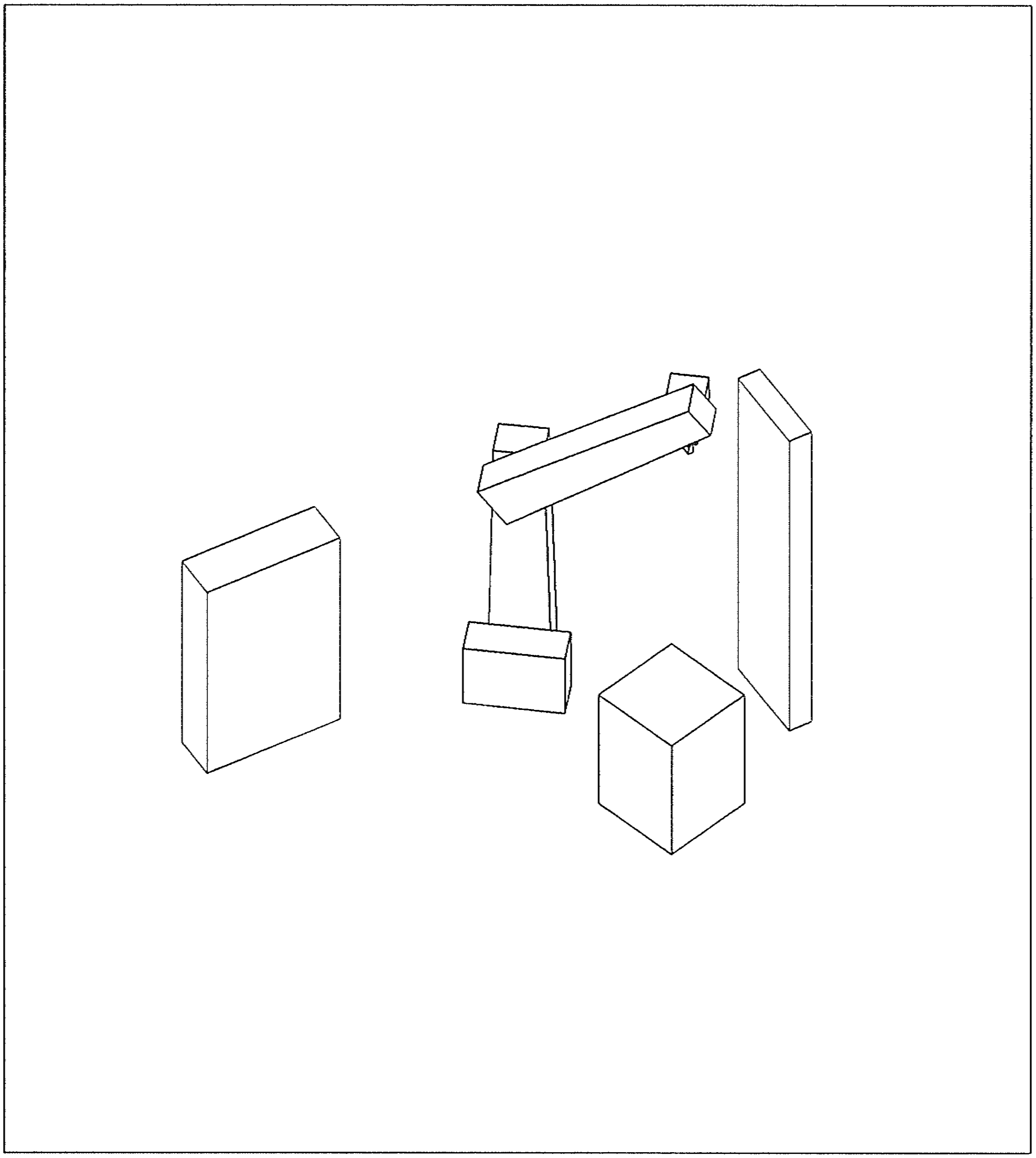
PATH 2 STEP 9
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



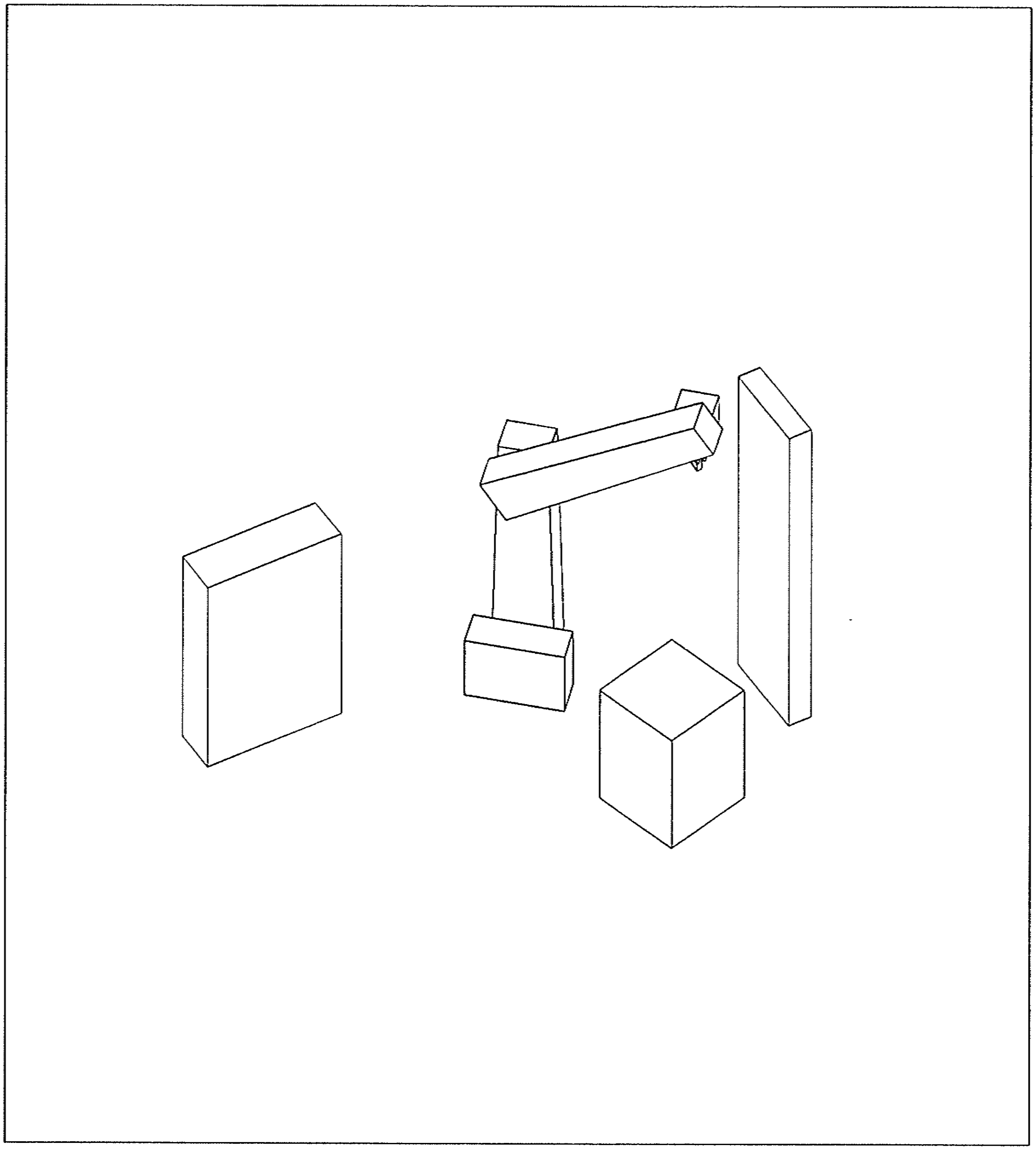
PATH 2 STEP 10
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



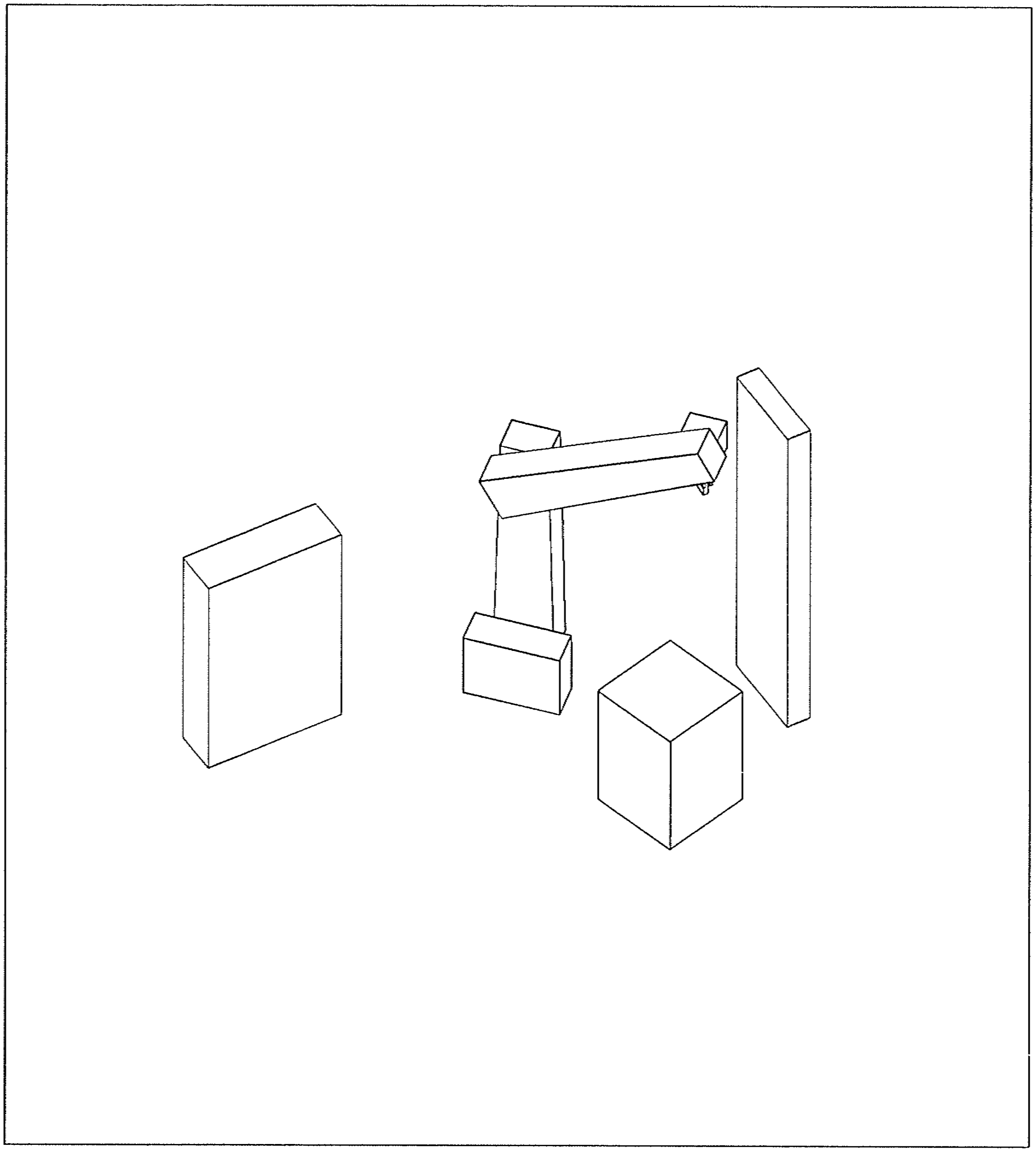
PATH 2 STEP 11
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



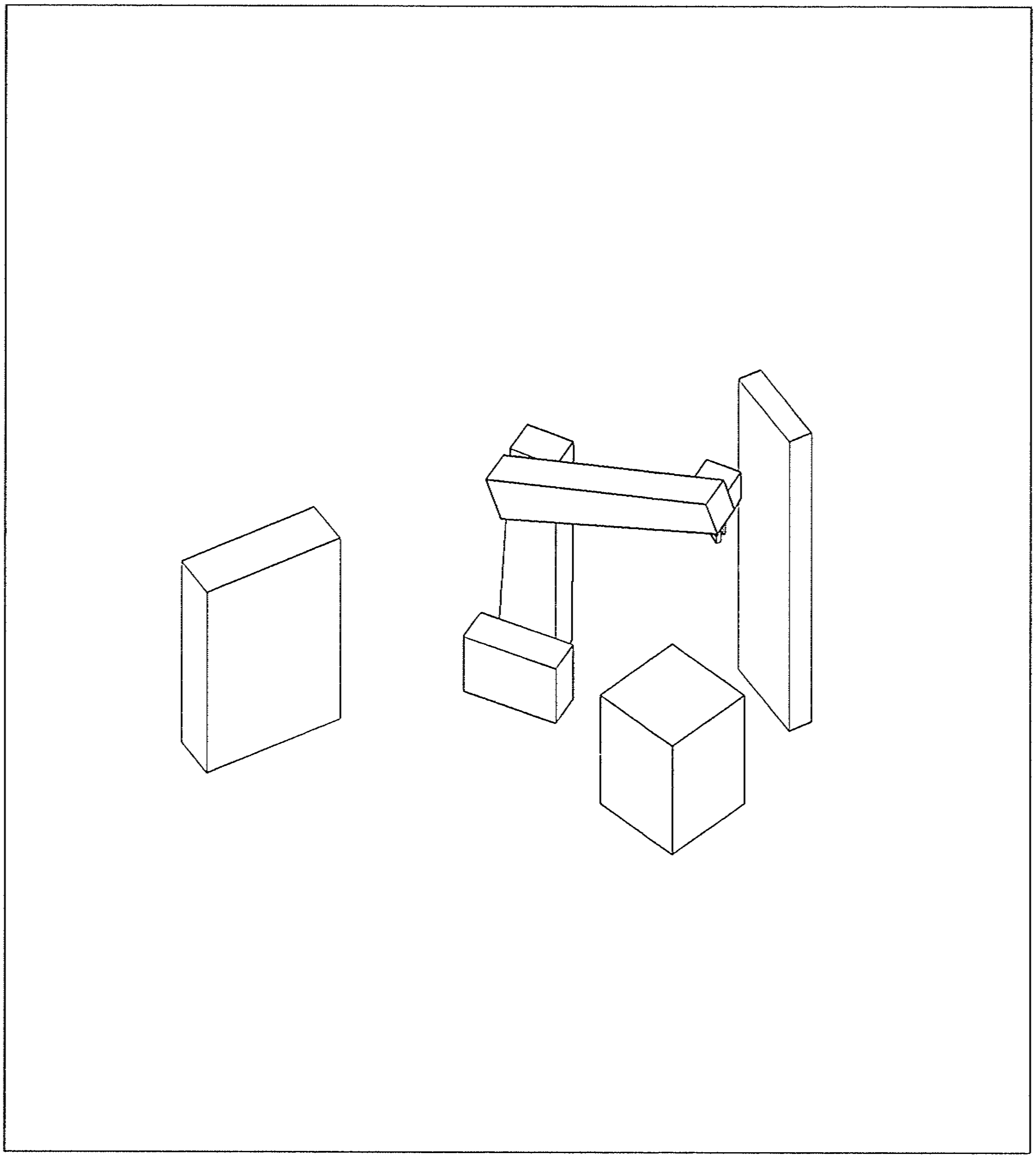
PATH 2 STEP 12
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



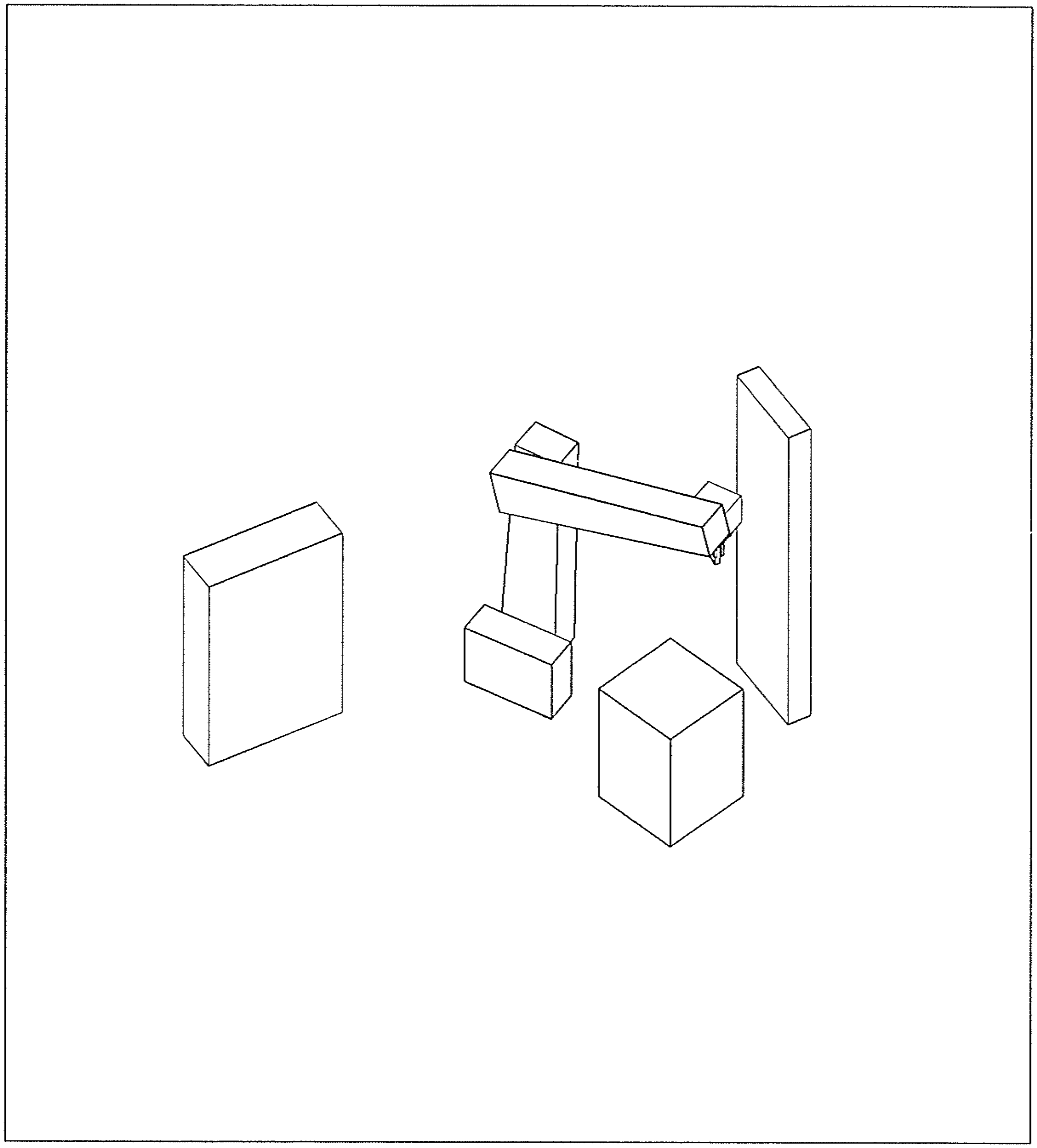
PATH 2 STEP 13
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



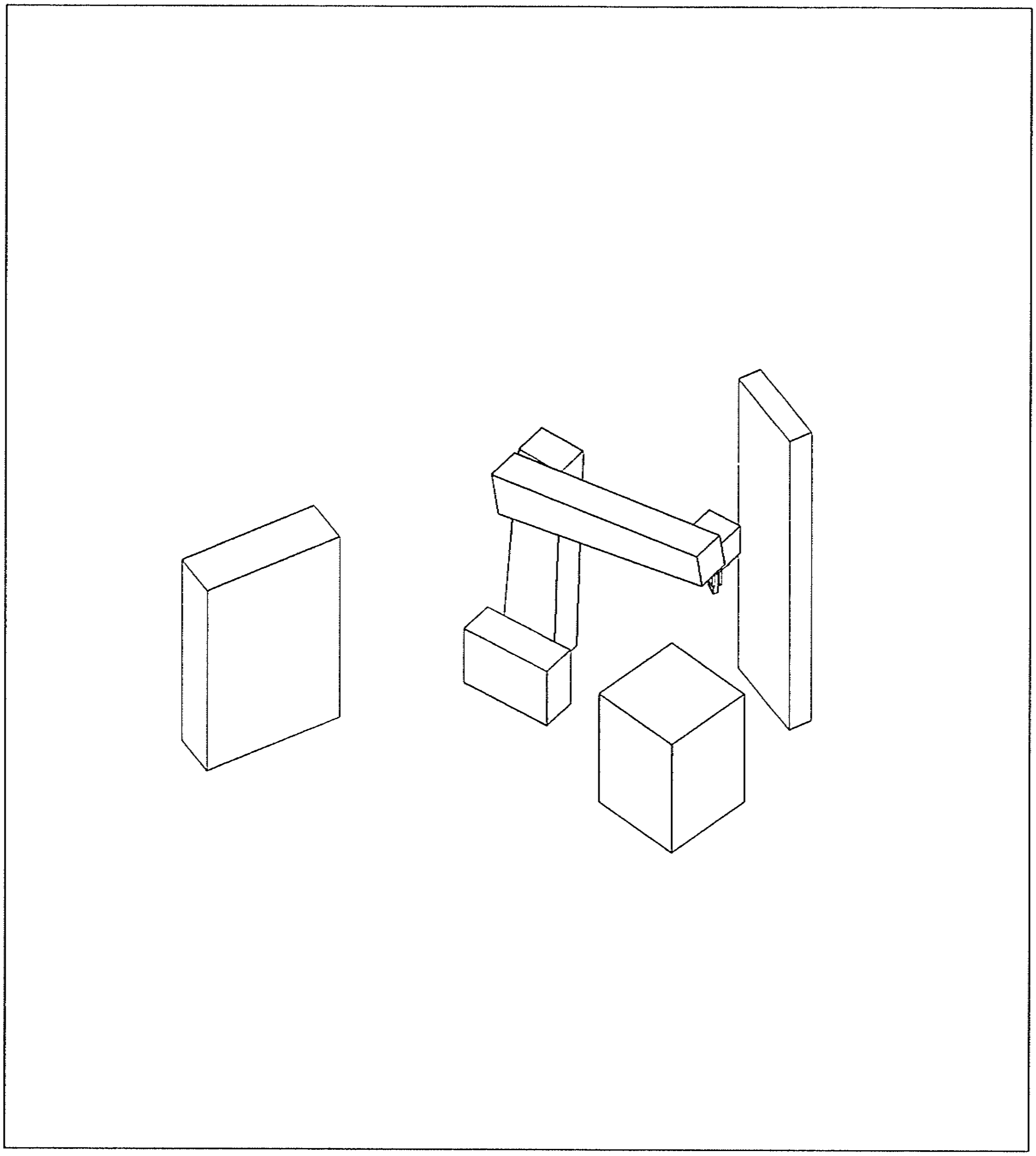
PATH 2 STEP 14
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



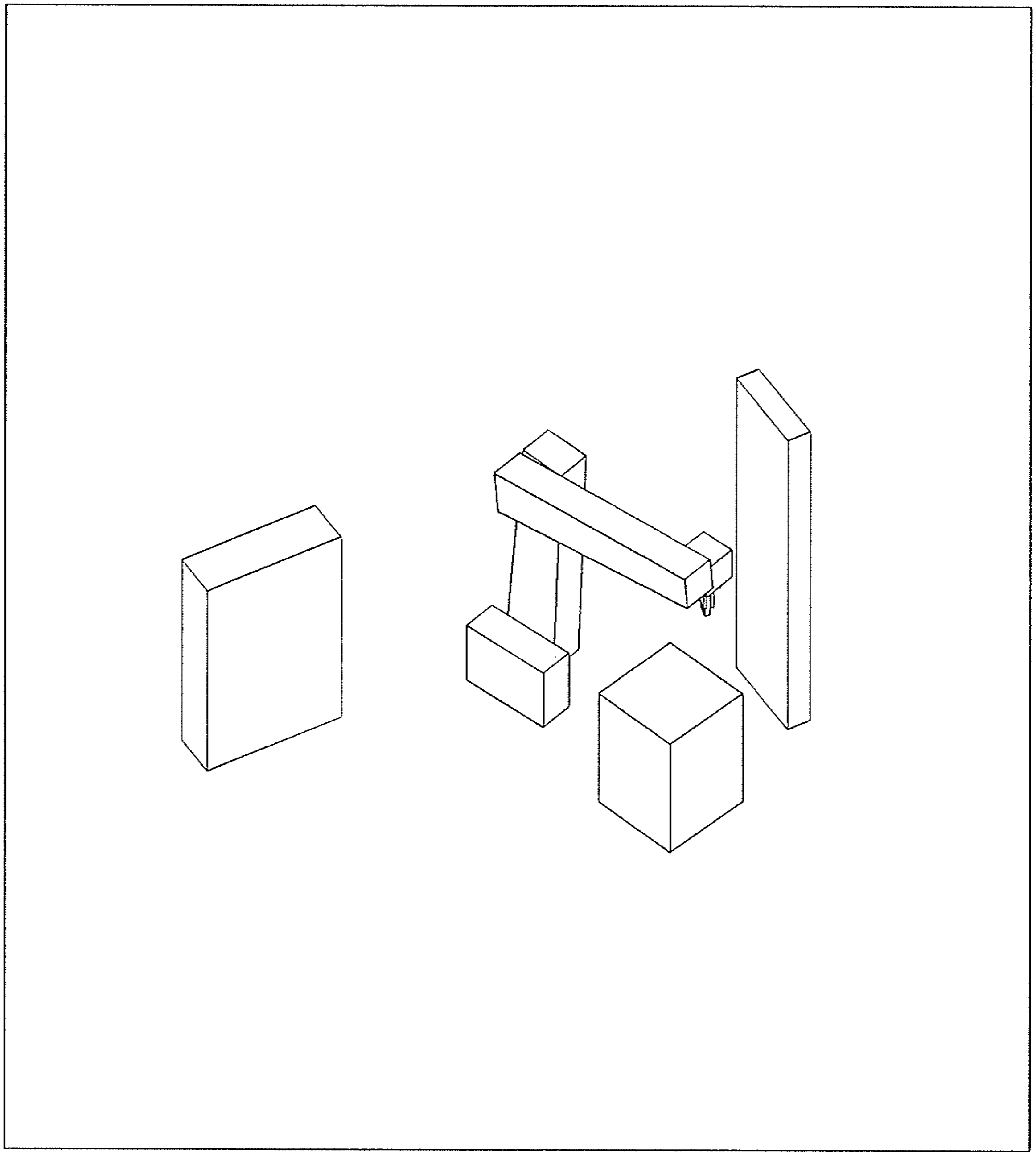
PATH 2 STEP 15
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



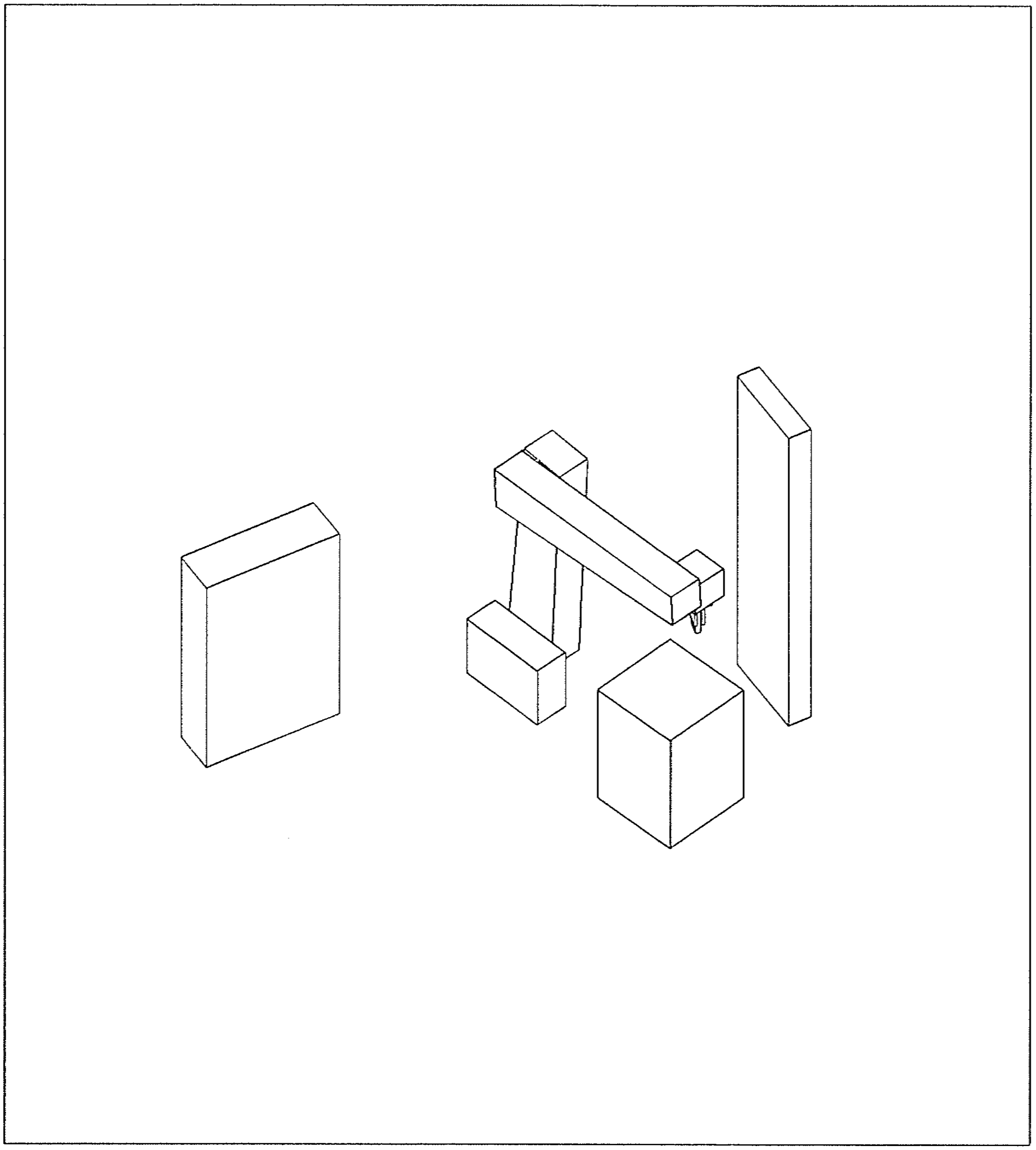
PATH 2 STEP 16
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



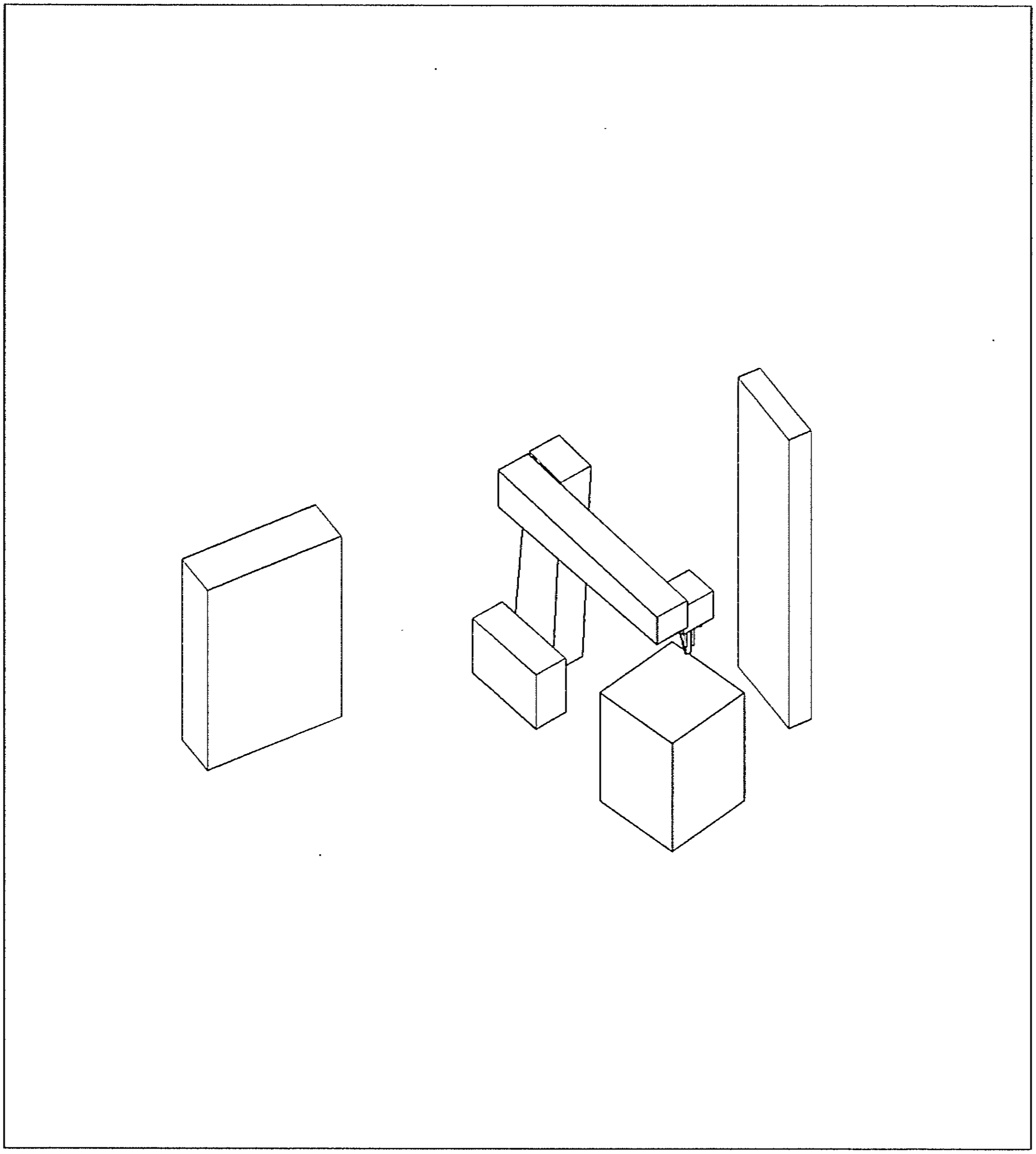
PATH 2 STEP 17
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



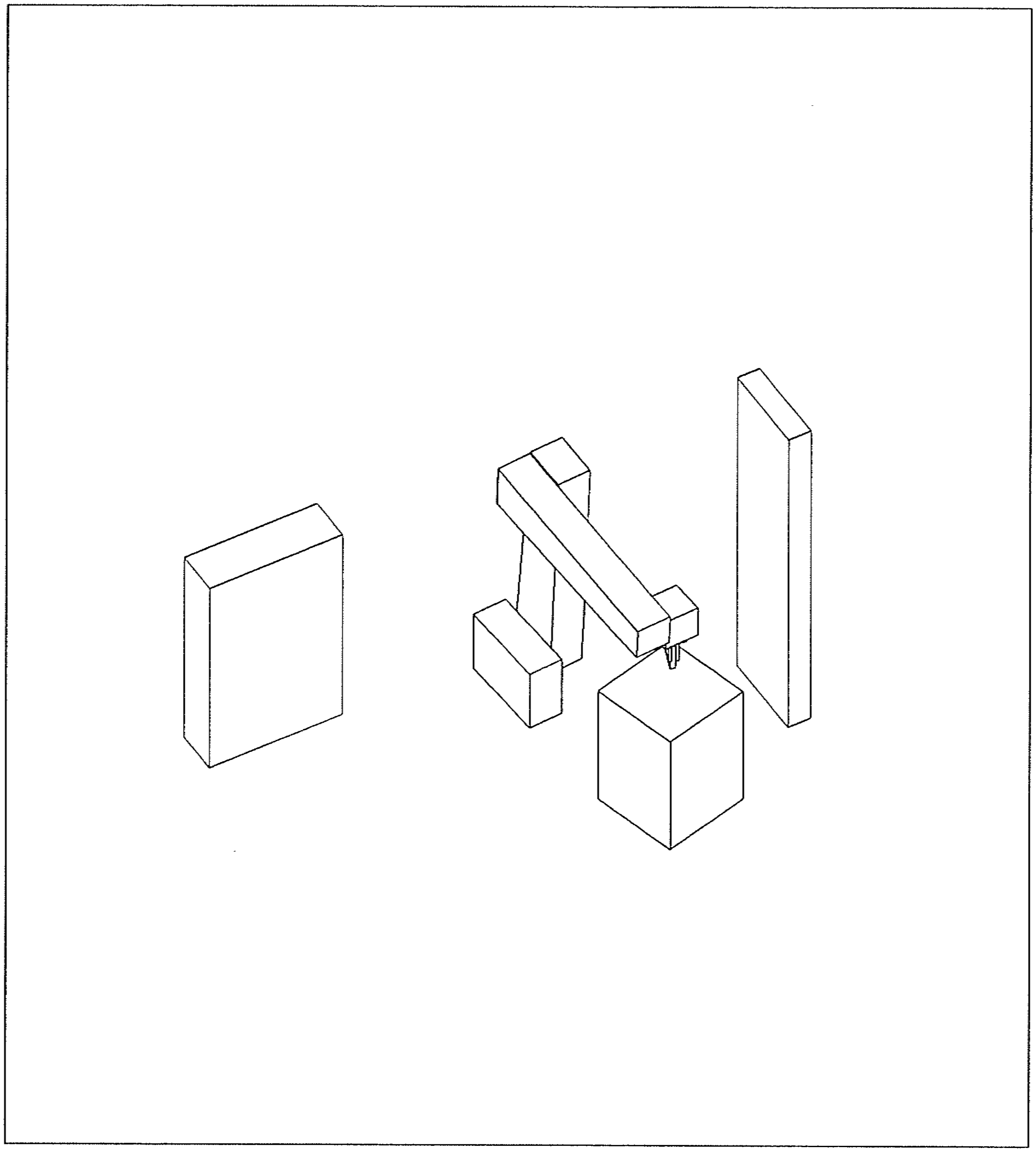
PATH 2 STEP 18
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



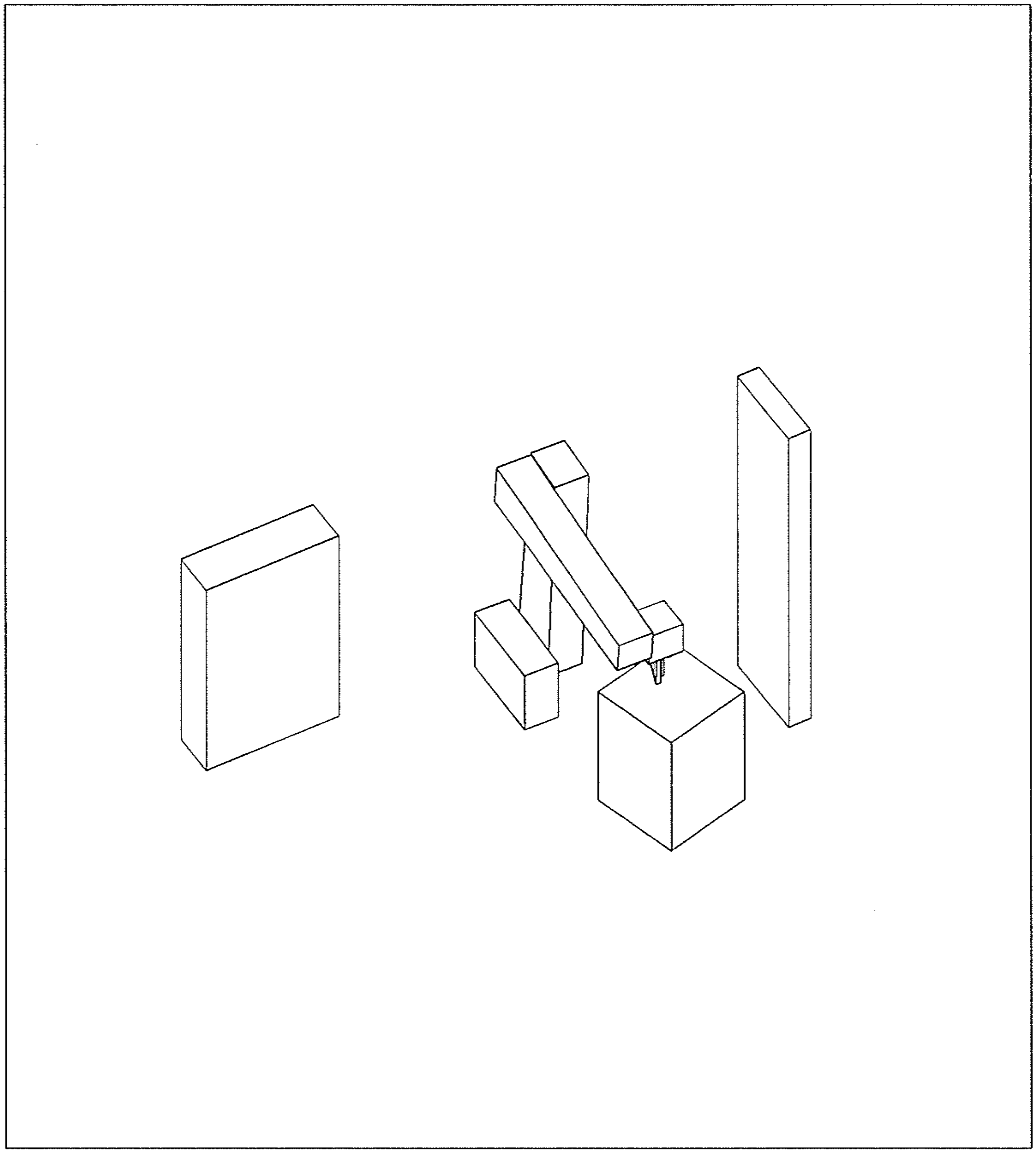
PATH 2 STEP 19
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



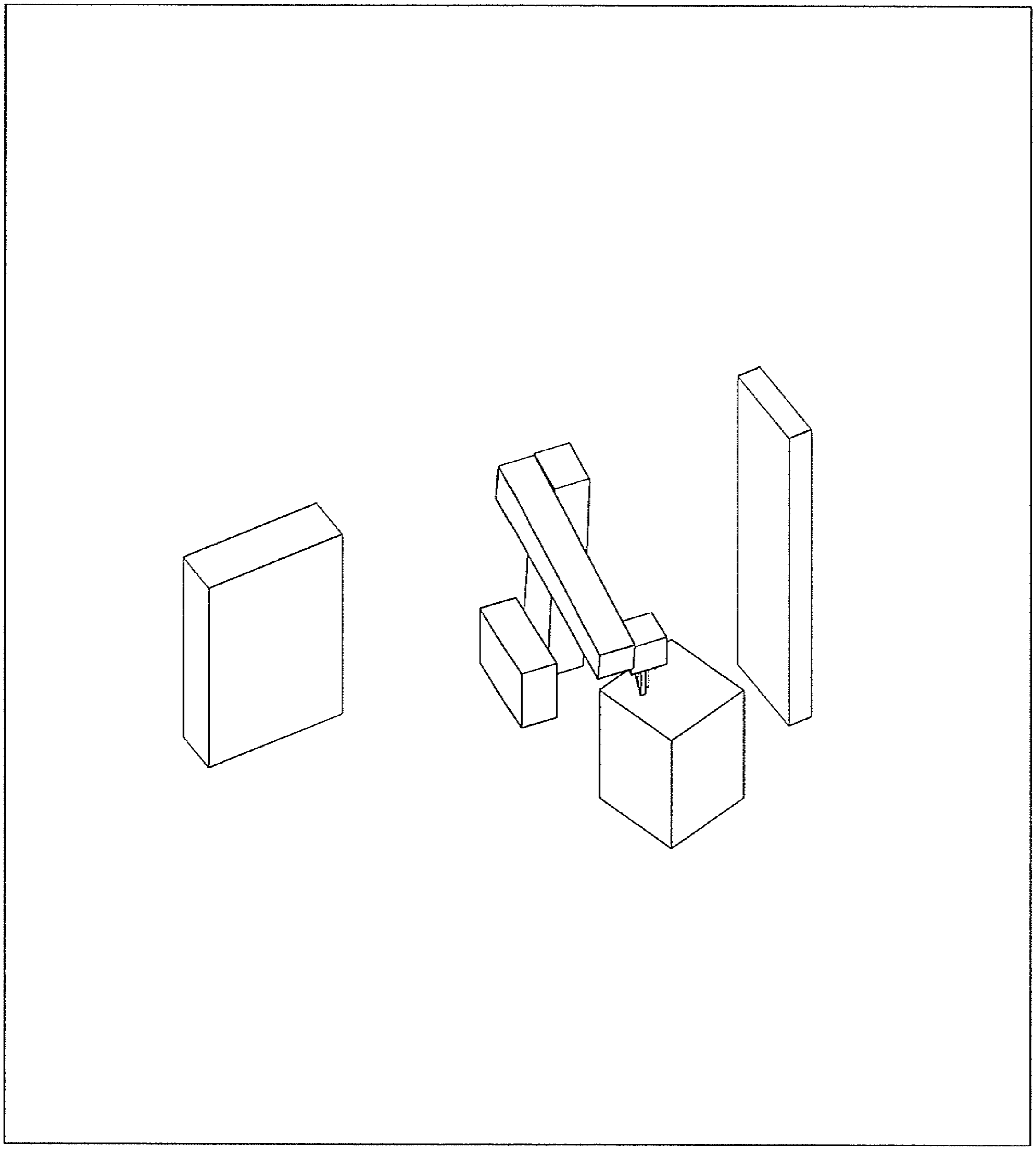
PATH 2 STEP 20
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



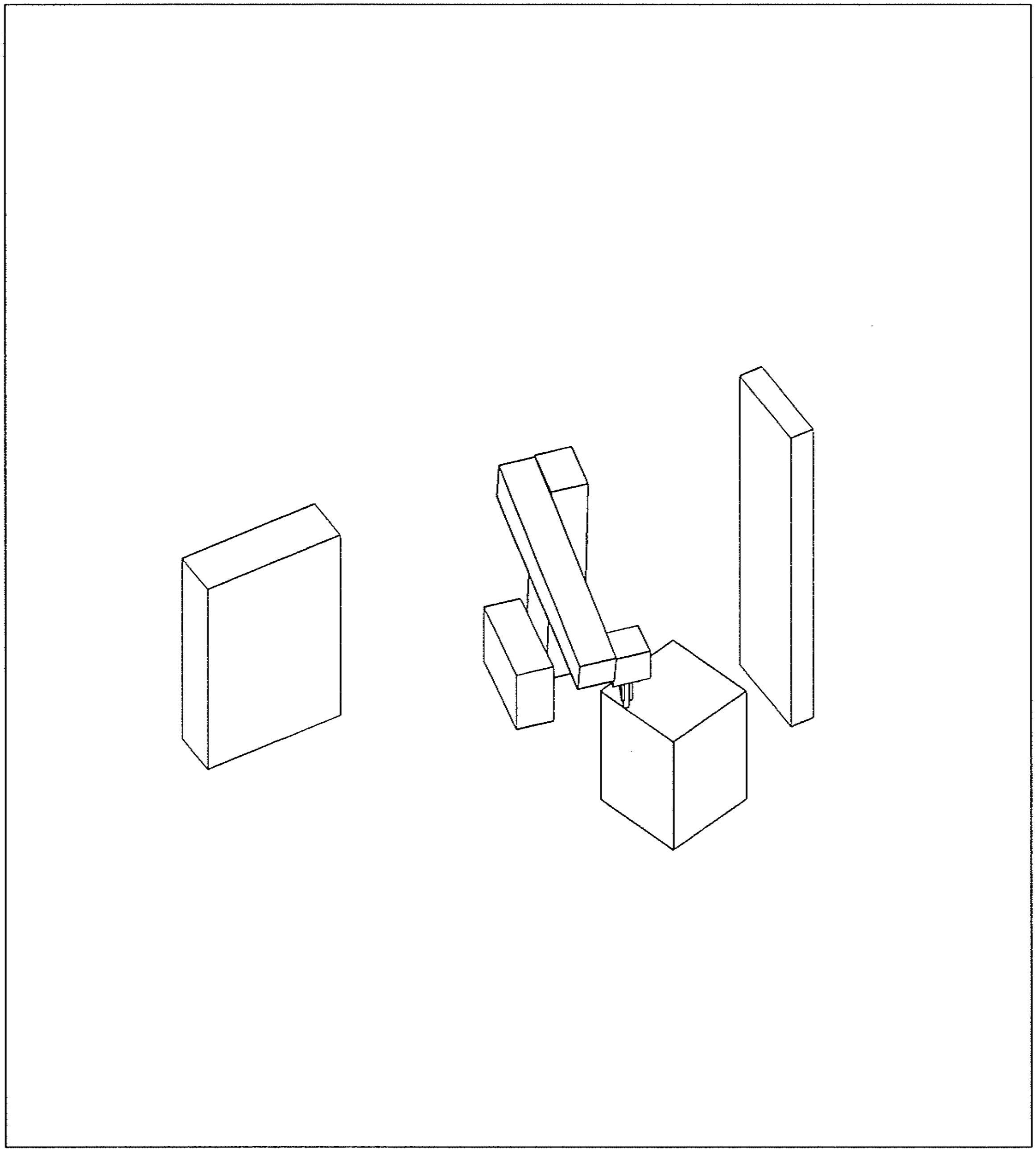
PATH 2 STEP 21
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



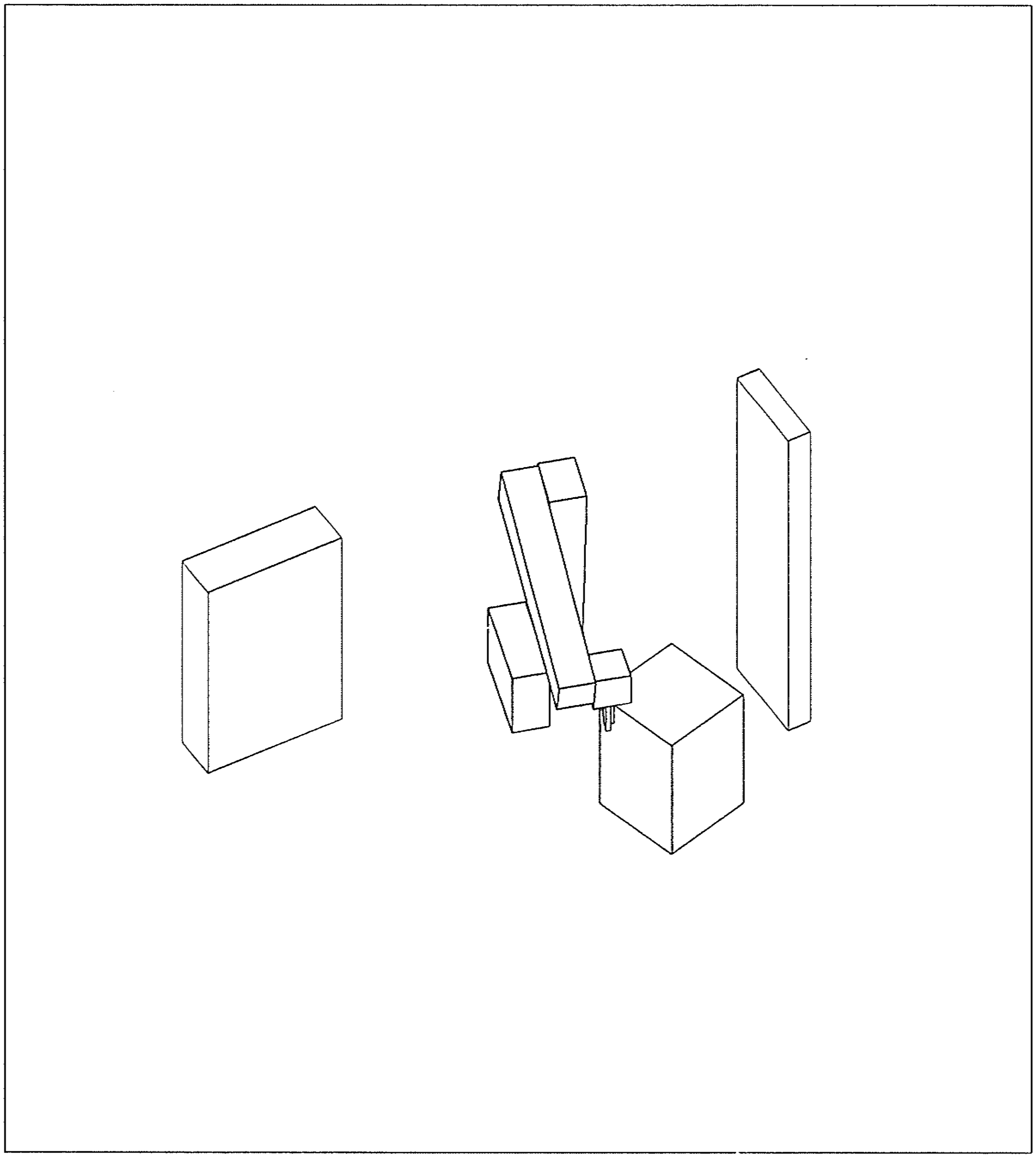
PATH 2 STEP 22
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



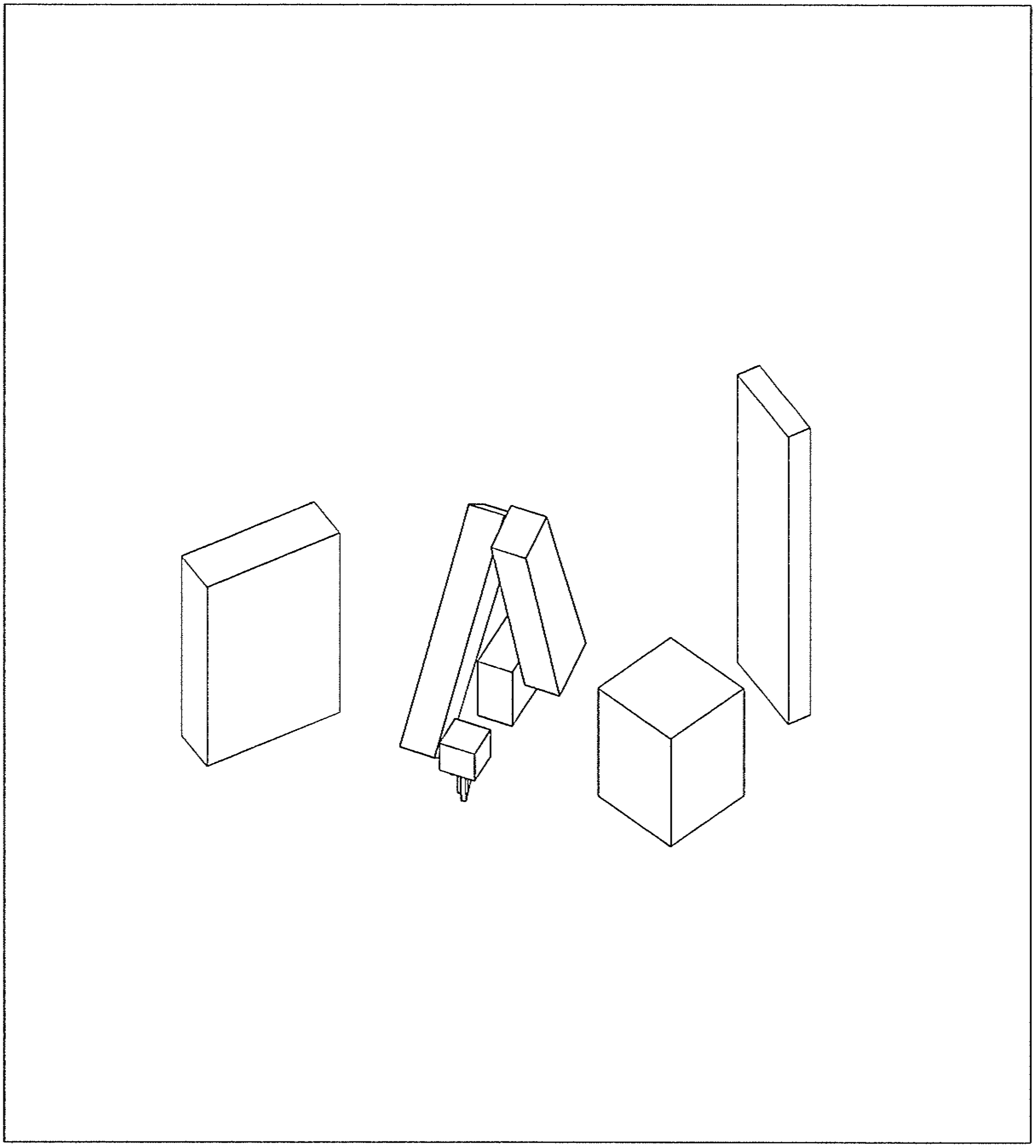
PATH 2 STEP 23
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



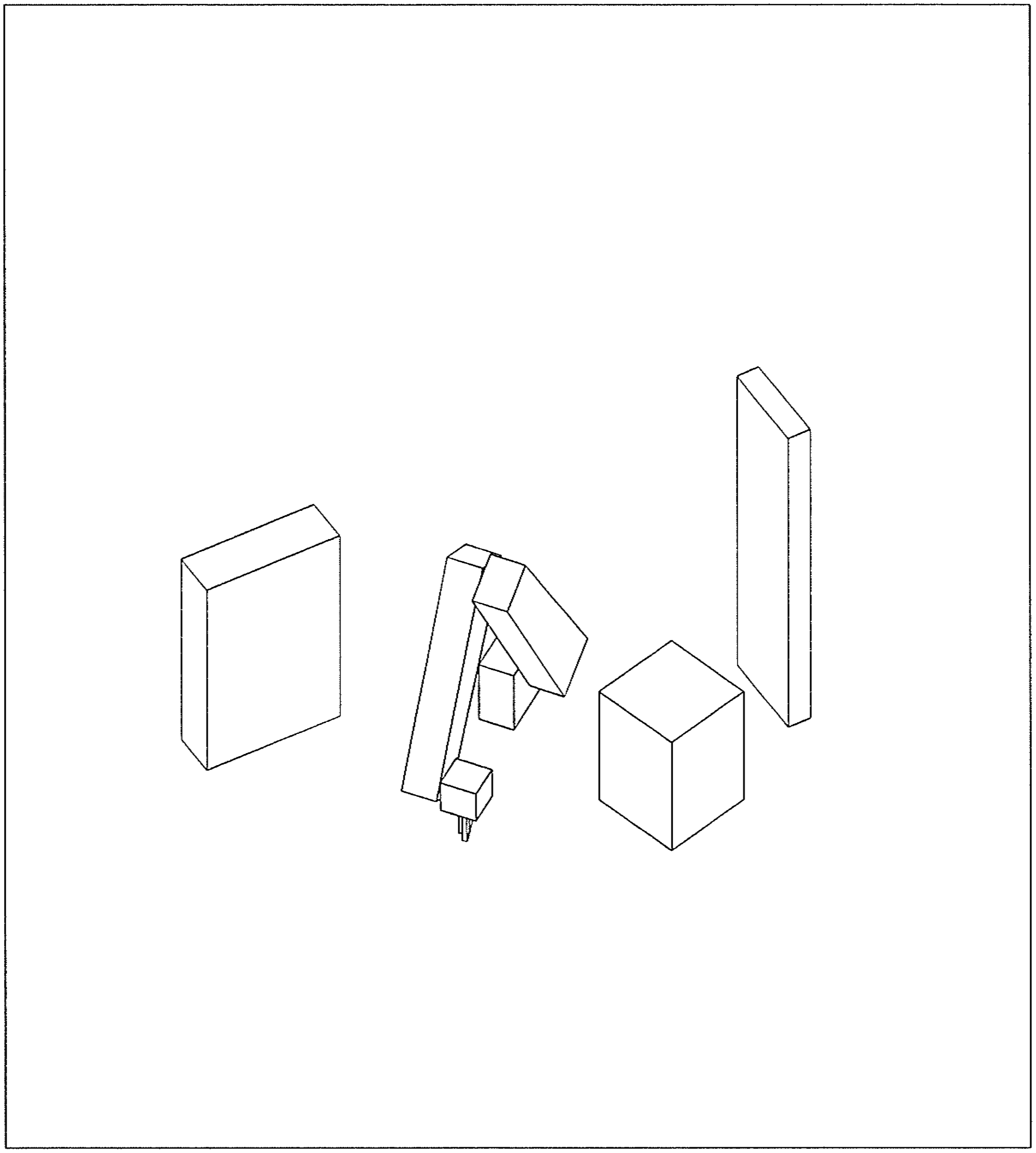
PATH 2 STEP 24
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



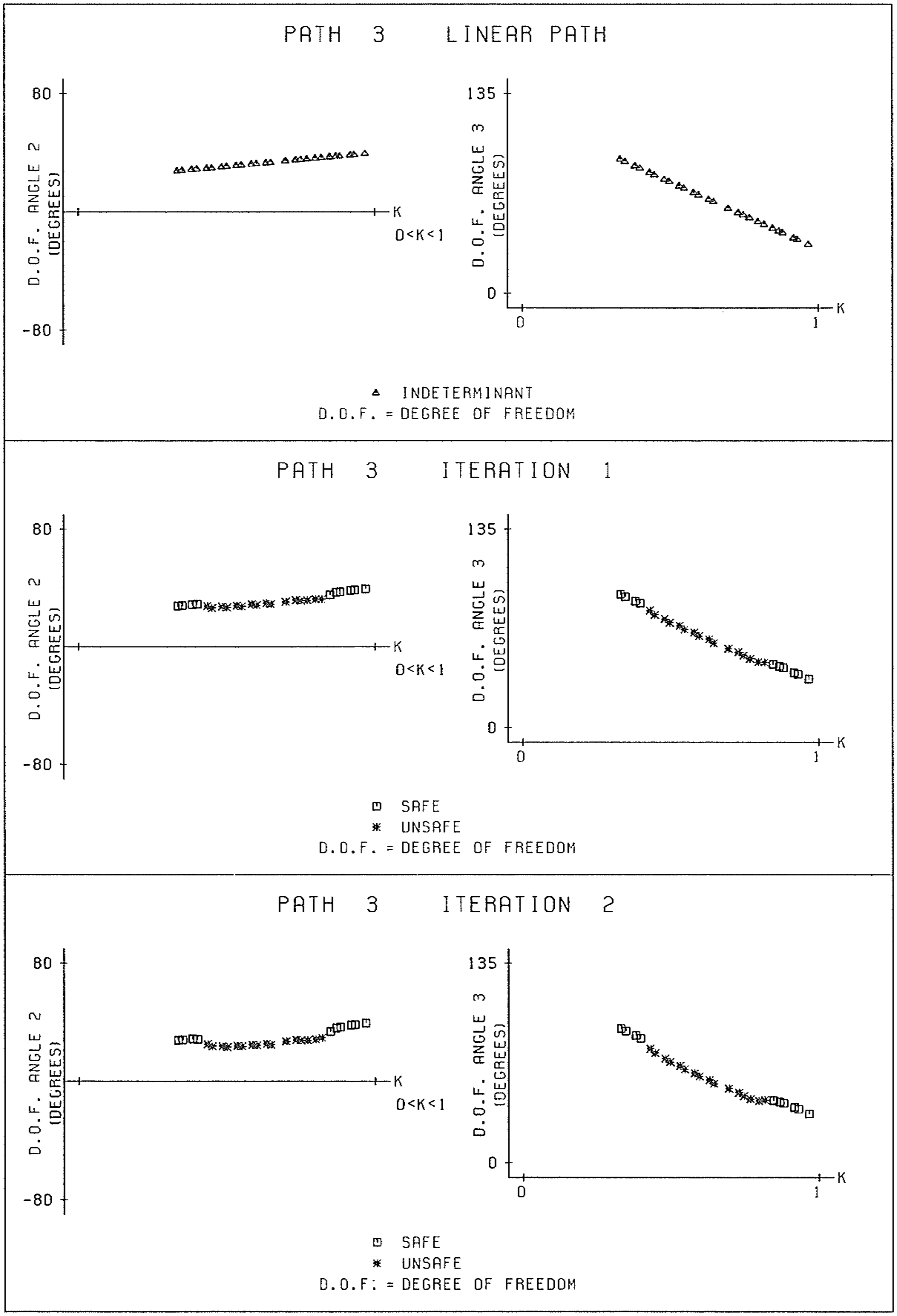
PATH 2 STEP 25
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



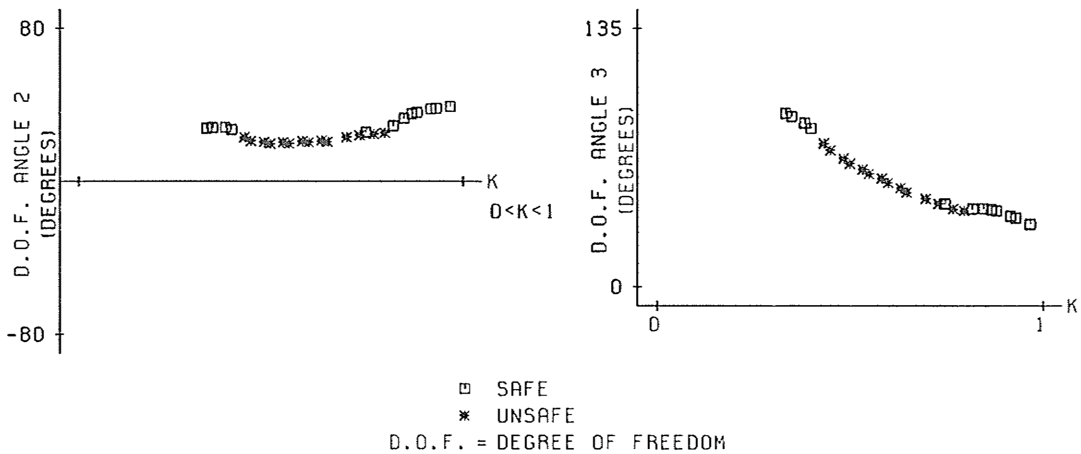
PATH 2 STEP 26
SAFE FIXED CONFIGURATION
PATH IS SAFE



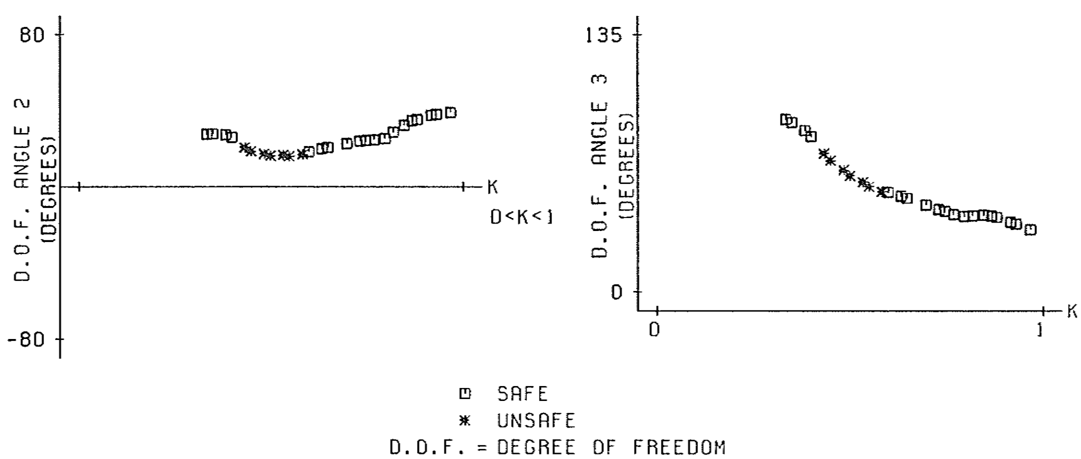
PATH 2 STEP 27
SAFE FIXED CONFIGURATION
PATH IS SAFE


PATH 3 LINEAR PATH
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
K
0
1
INDETERMINANT
D.O.F. = DEGREE OF FREEDOM
PATH 3 ITERATION 1
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
K
0
1
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM
PATH 3 ITERATION 2
80
-80
D.O.F. ANGLE 2 (DEGREES)
K
0<K<1
135
0
D.O.F. ANGLE 3 (DEGREES)
K
0
1
SAFE
UNSAFE
D.O.F. = DEGREE OF FREEDOM

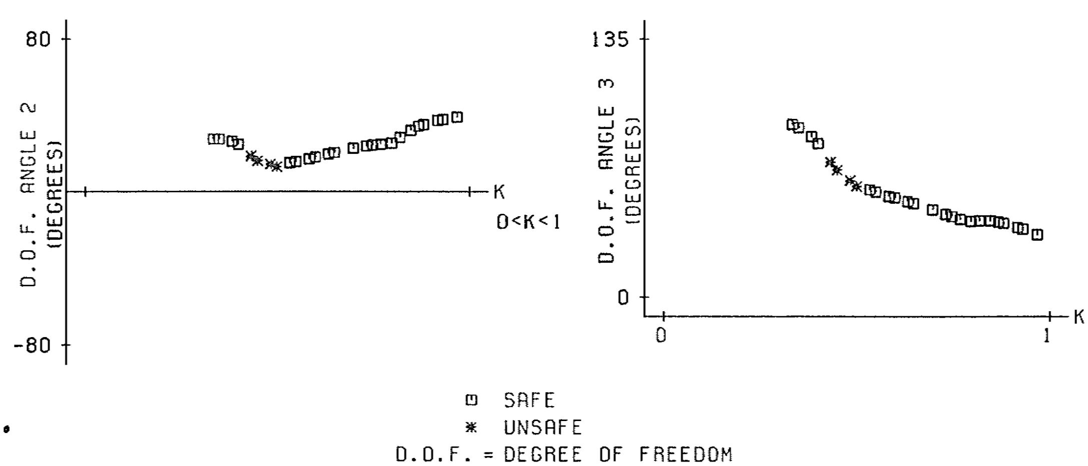
PATH 3 ITERATION 3



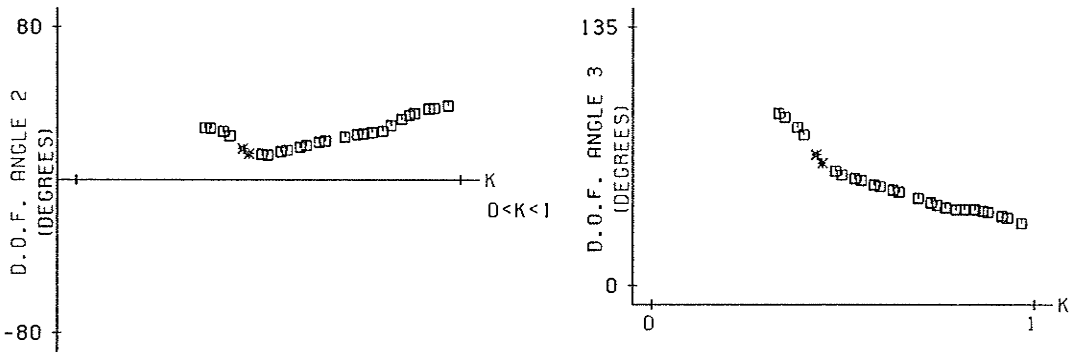
PATH 3 ITERATION 4



PATH 3 ITERATION 5

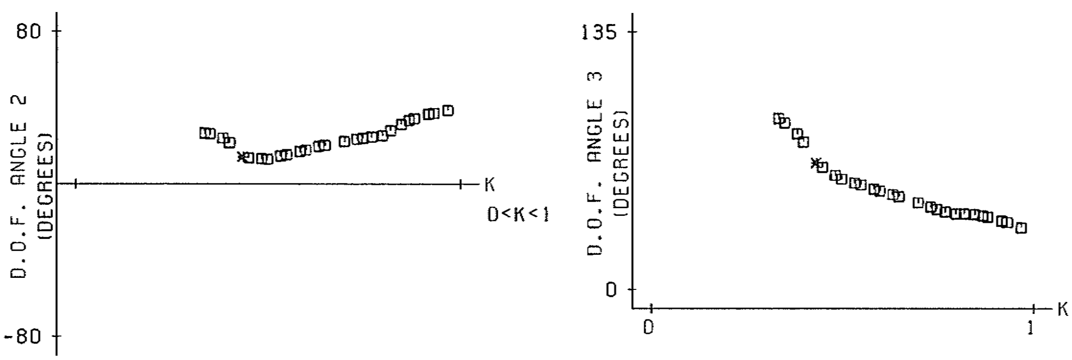


PATH 3 ITERATION 6



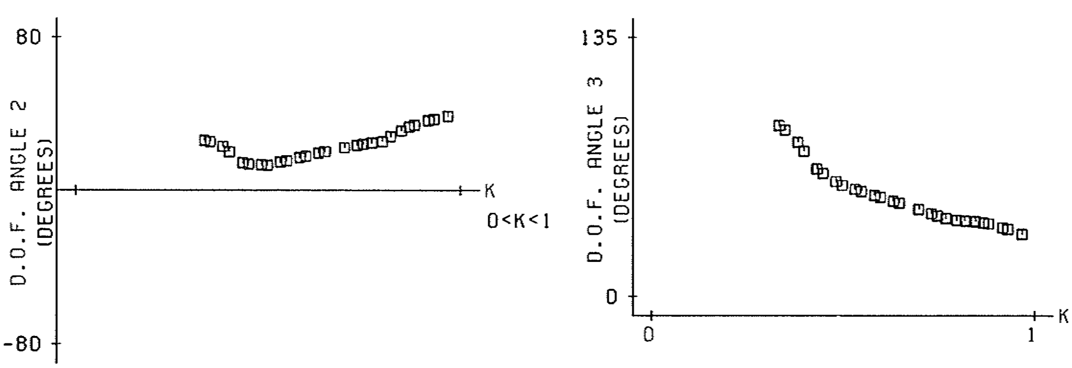
□ SAFE
* UNSAFE
D.O.F. = DEGREE OF FREEDOM

PATH 3 ITERATION 7

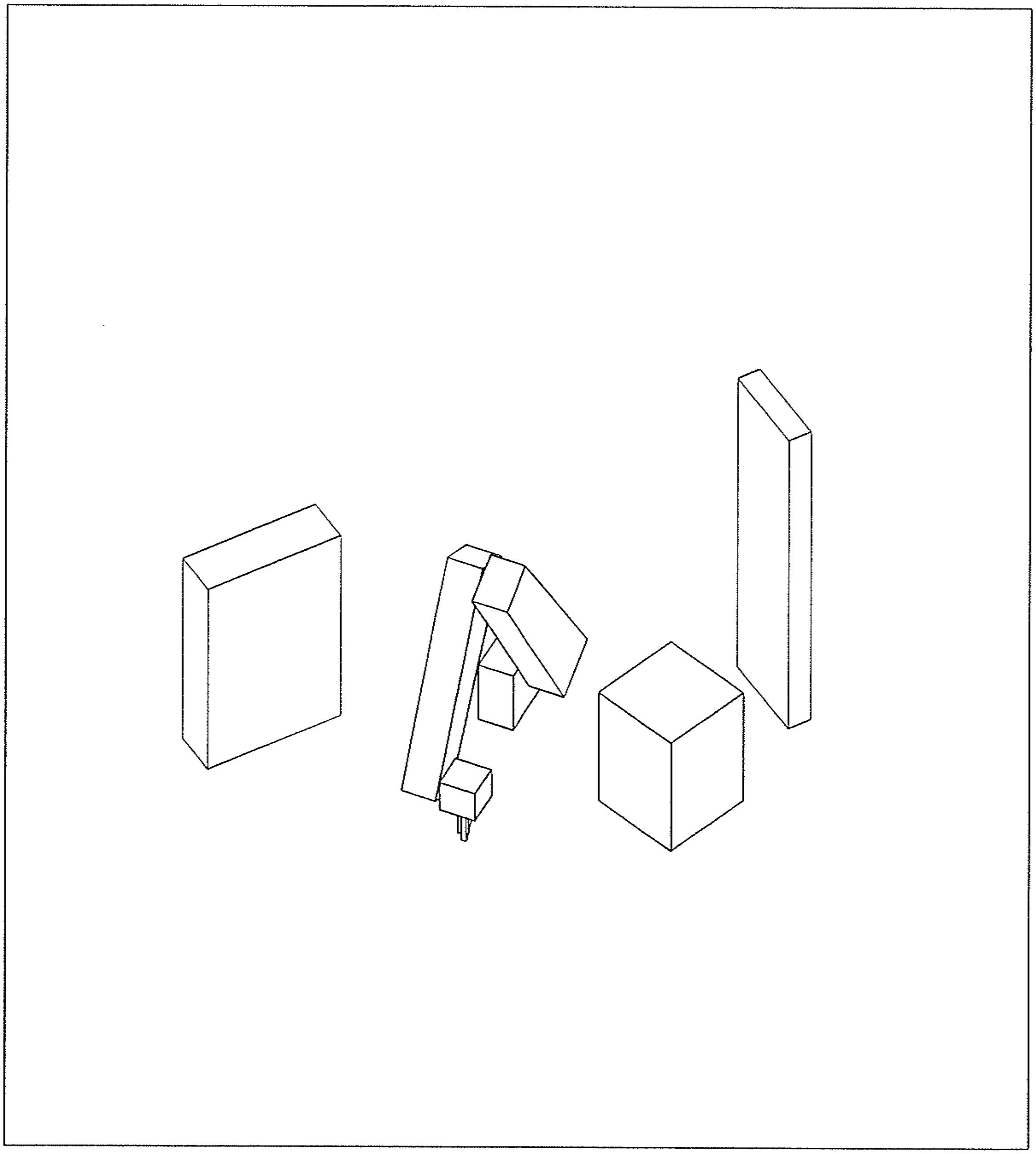


□ SAFE
* UNSAFE
D.O.F. = DEGREE OF FREEDOM

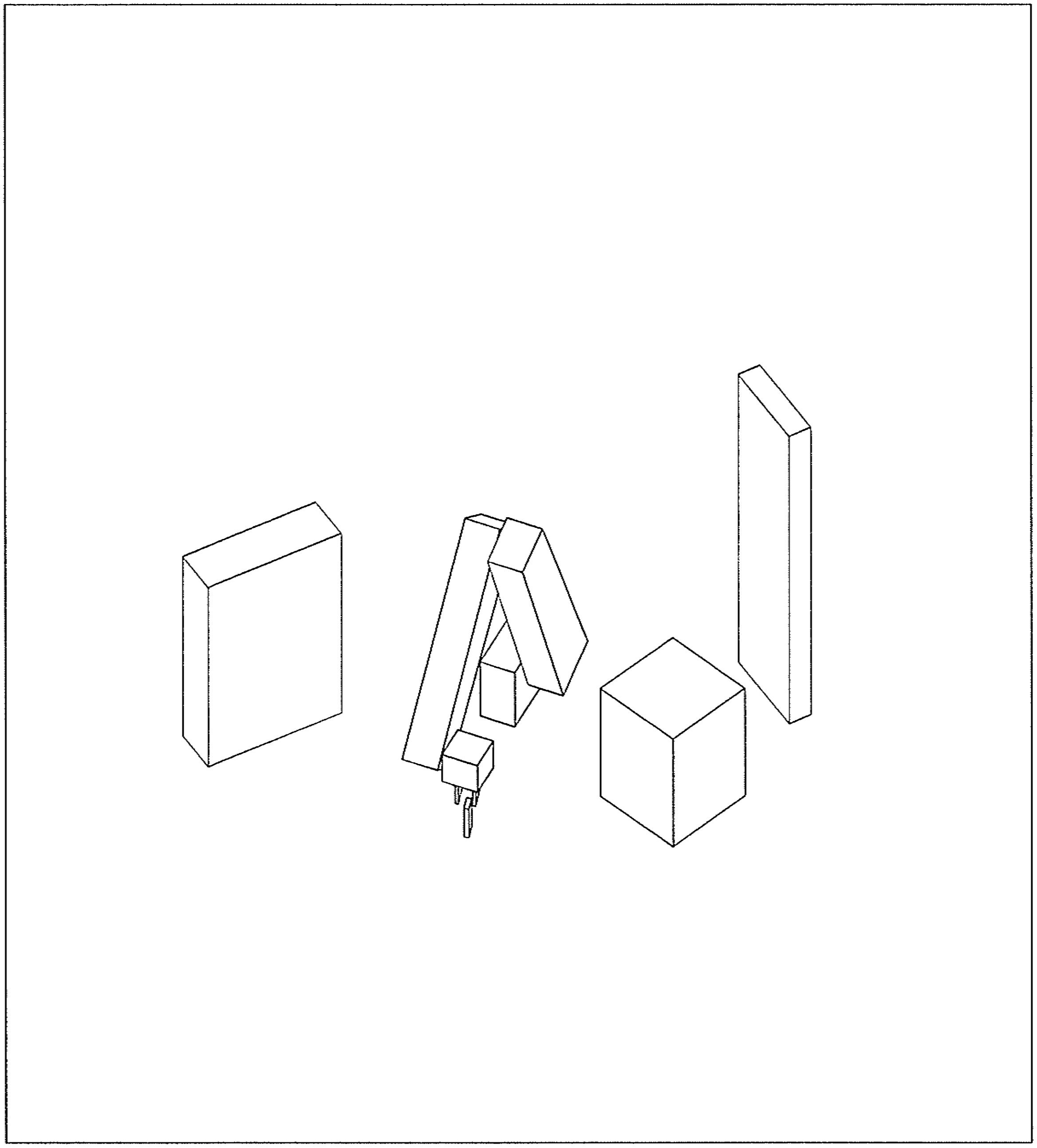
PATH 3 ITERATION 8



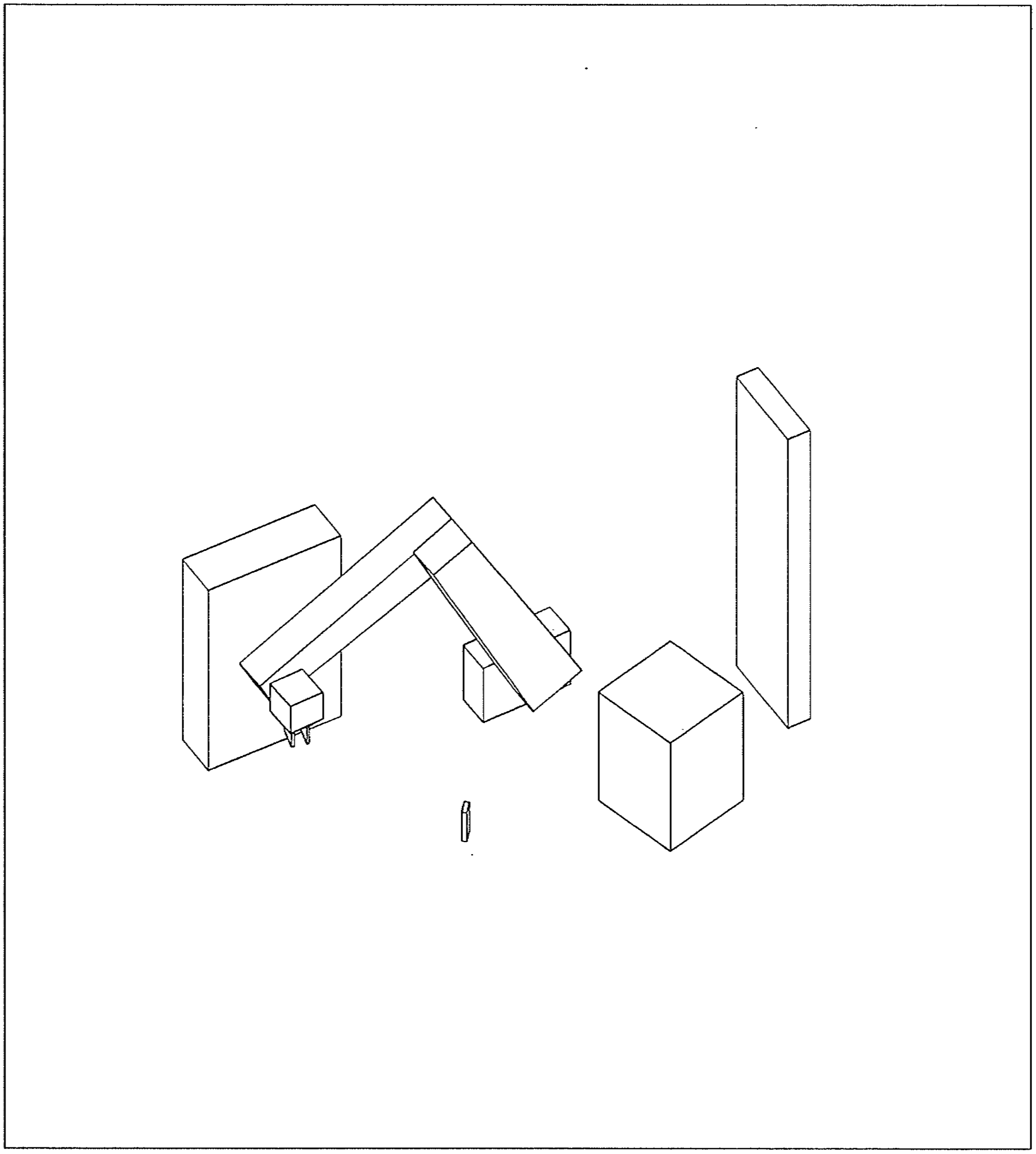
□ SAFE
* UNSAFE
D.O.F. = DEGREE OF FREEDOM



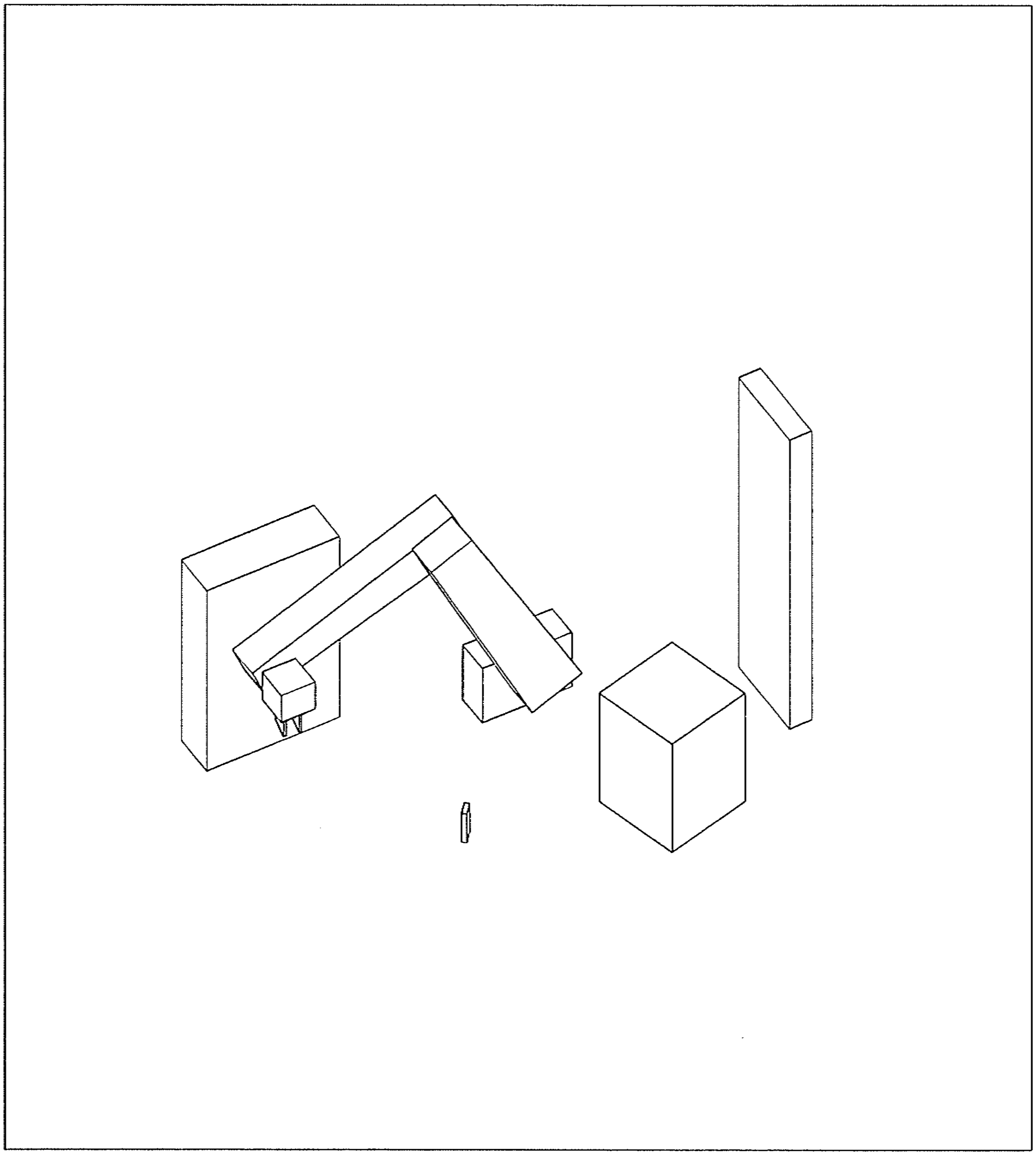
PATH 3 STEP 1
SAFE FIXED CONFIGURATION
PATH IS SAFE



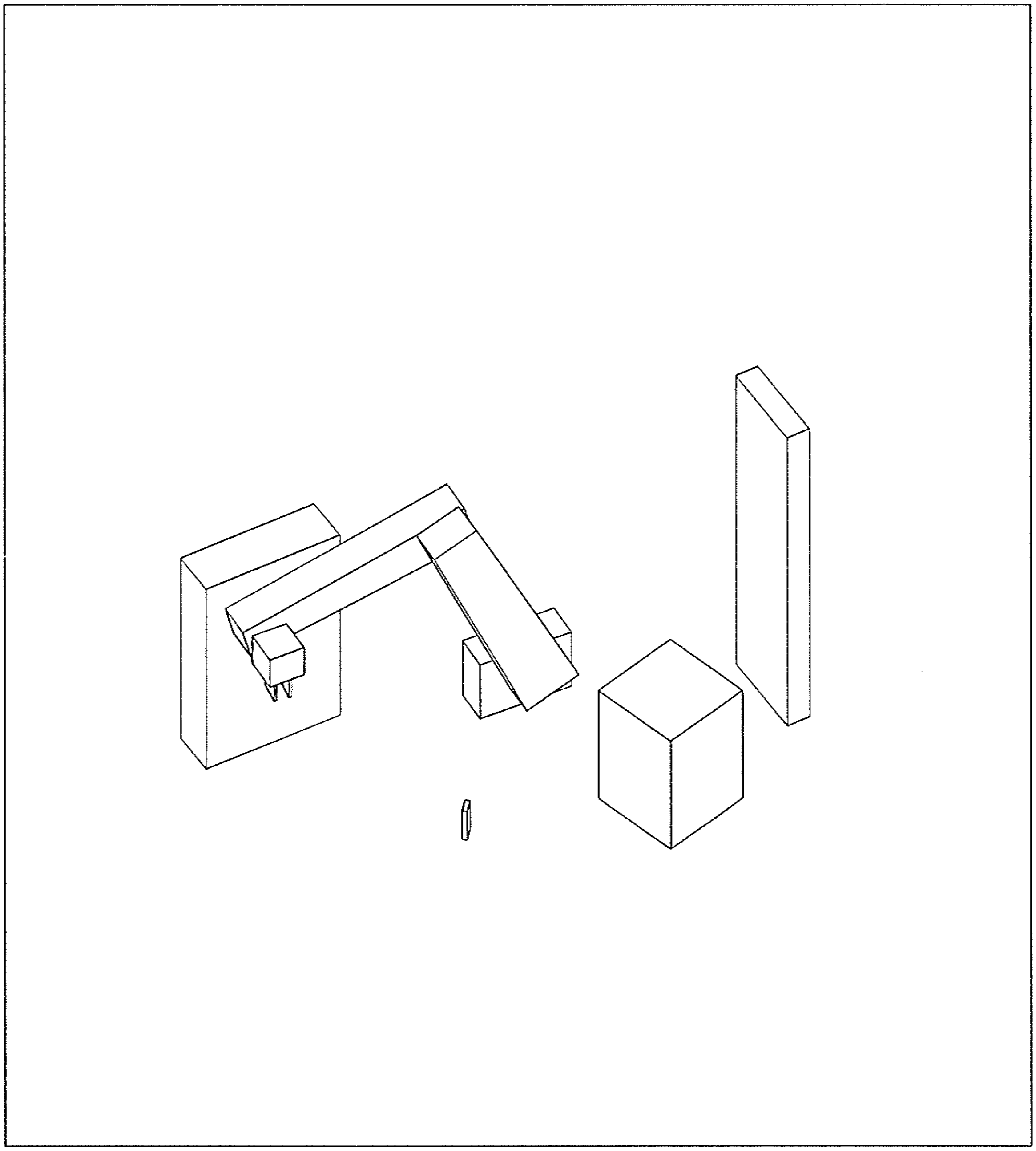
PATH 3 STEP 2
SAFE FIXED CONFIGURATION
PATH IS SAFE



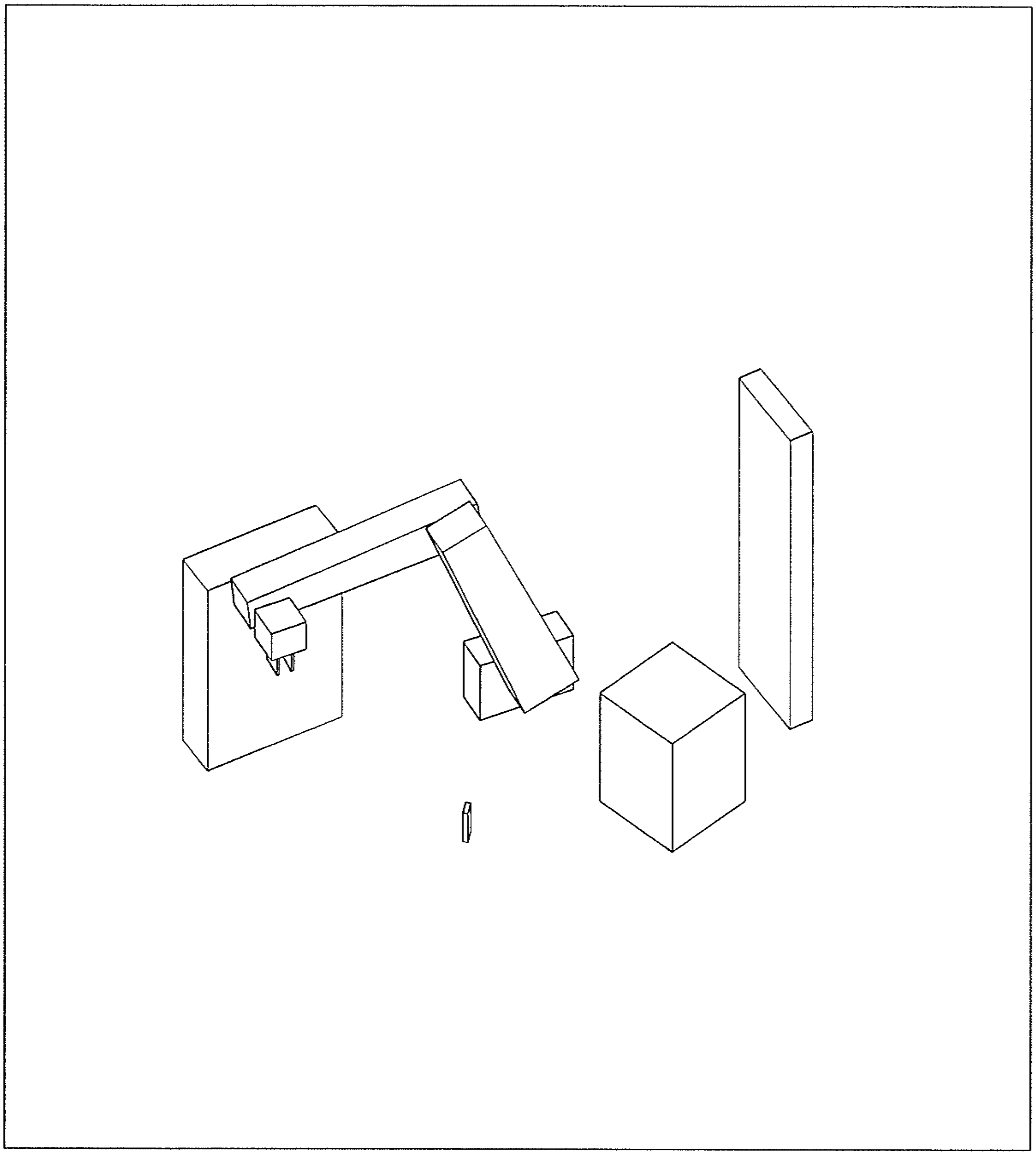
PATH 3 STEP 3
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



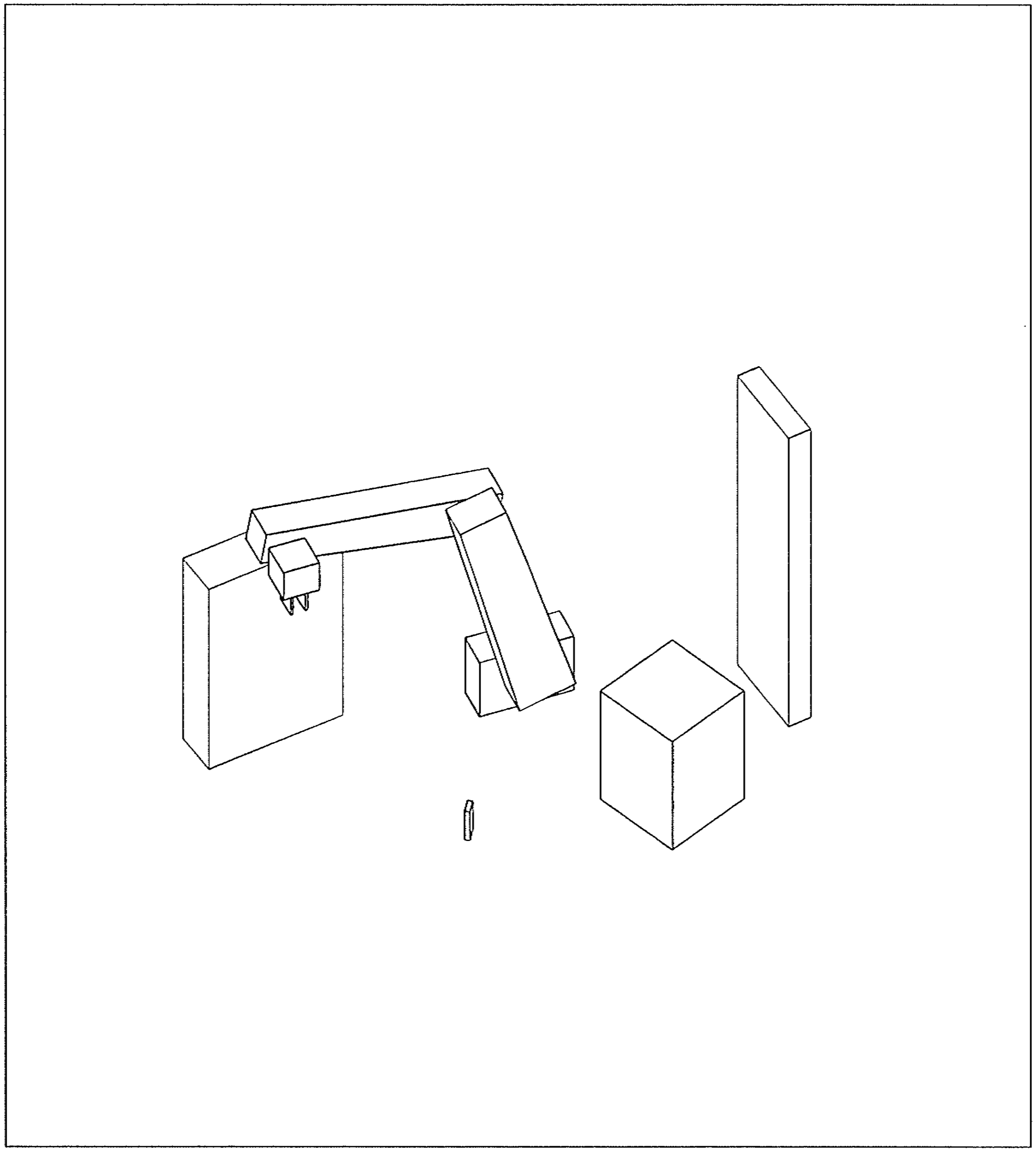
PATH 3 STEP 4
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



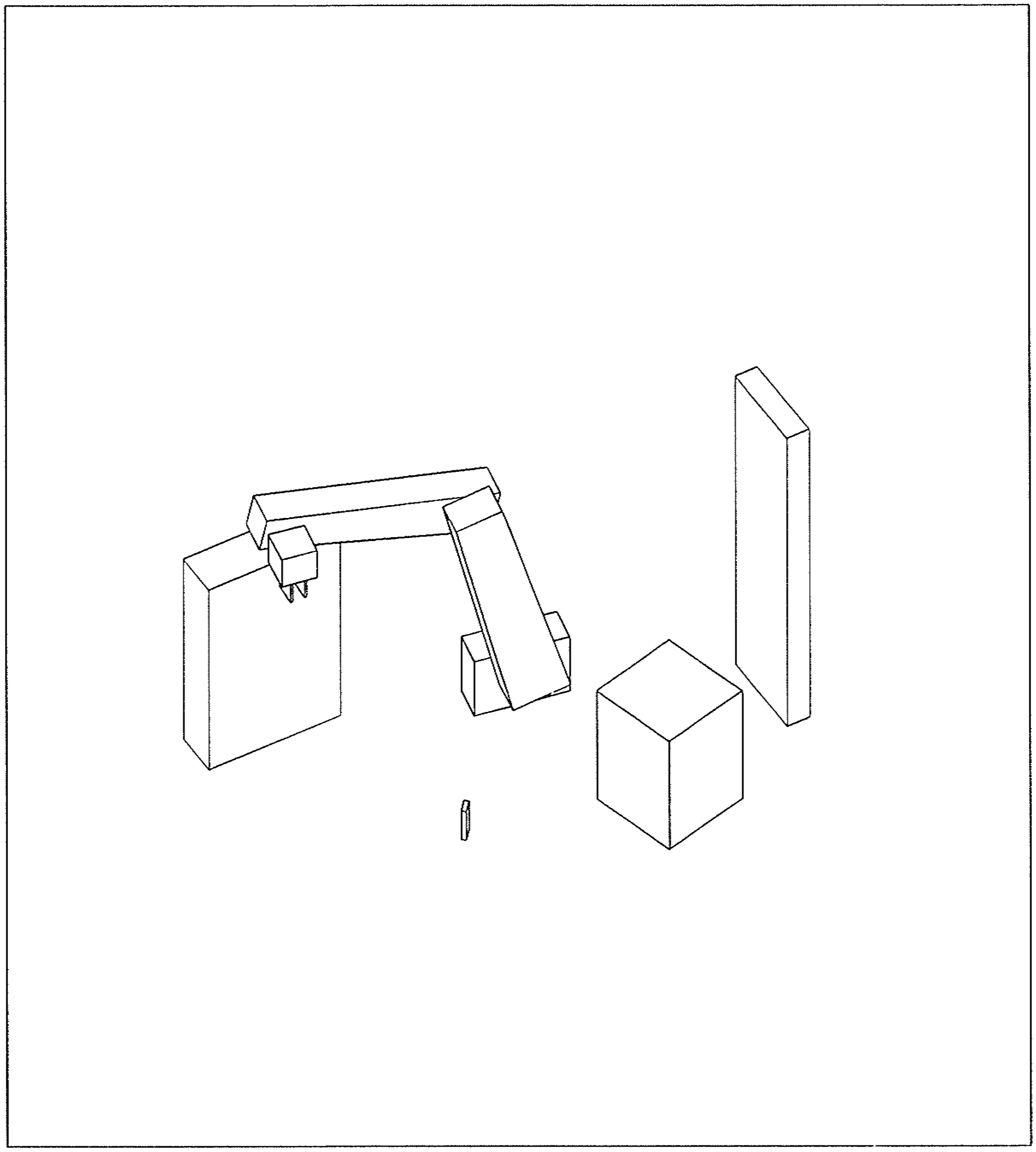
PATH 3 STEP 5
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



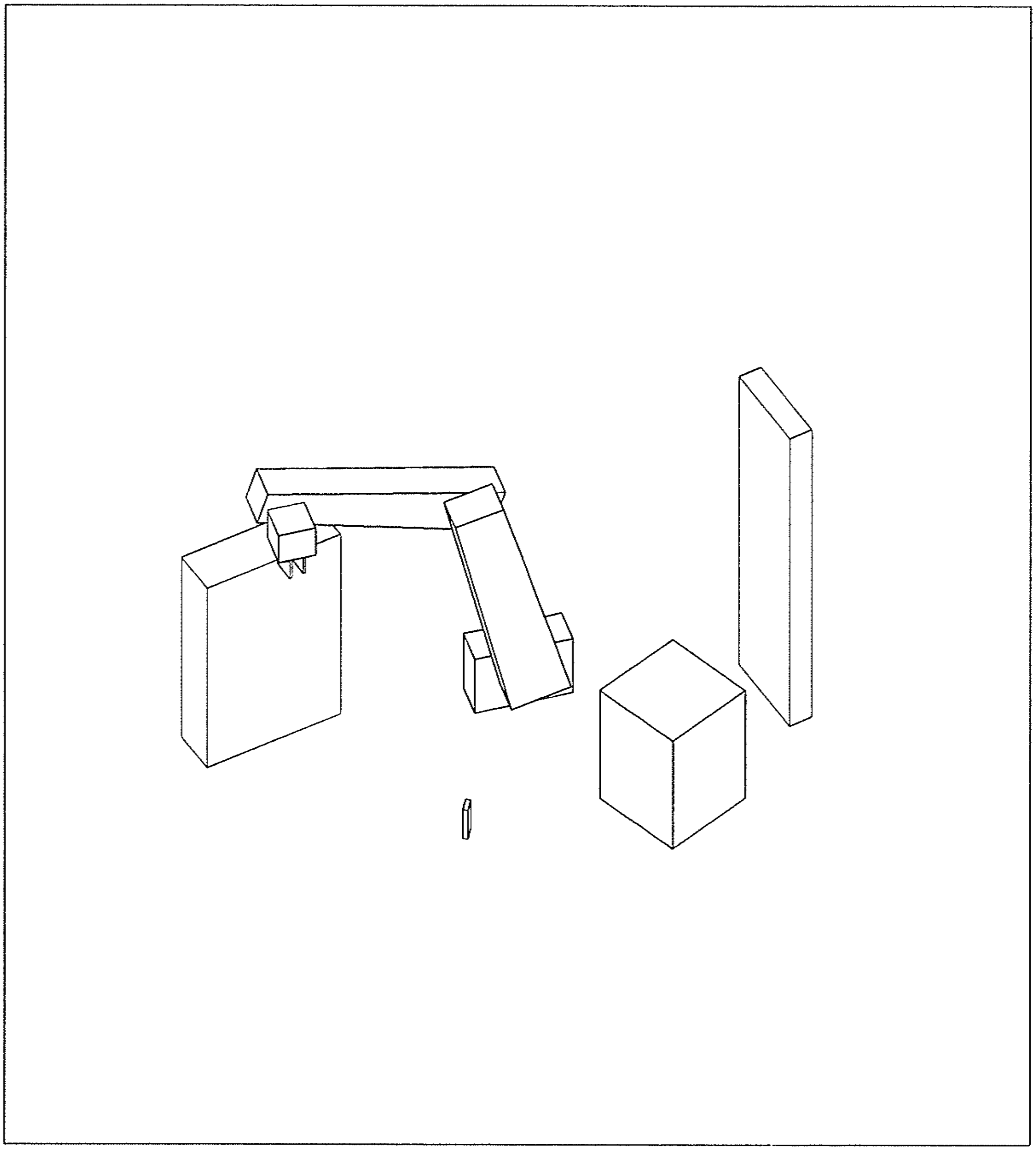
PATH 3 STEP 6
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



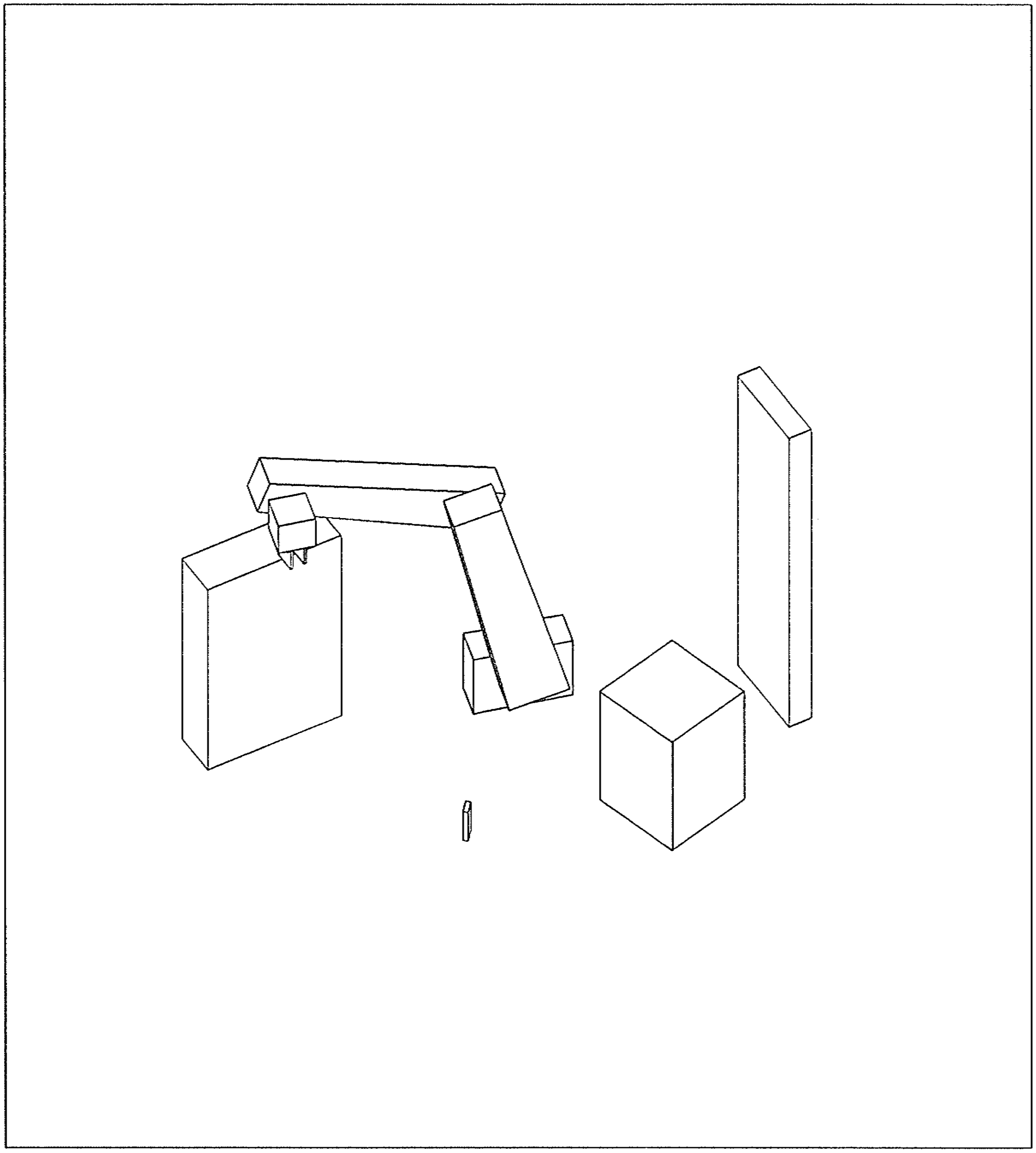
PATH 3 STEP 7
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



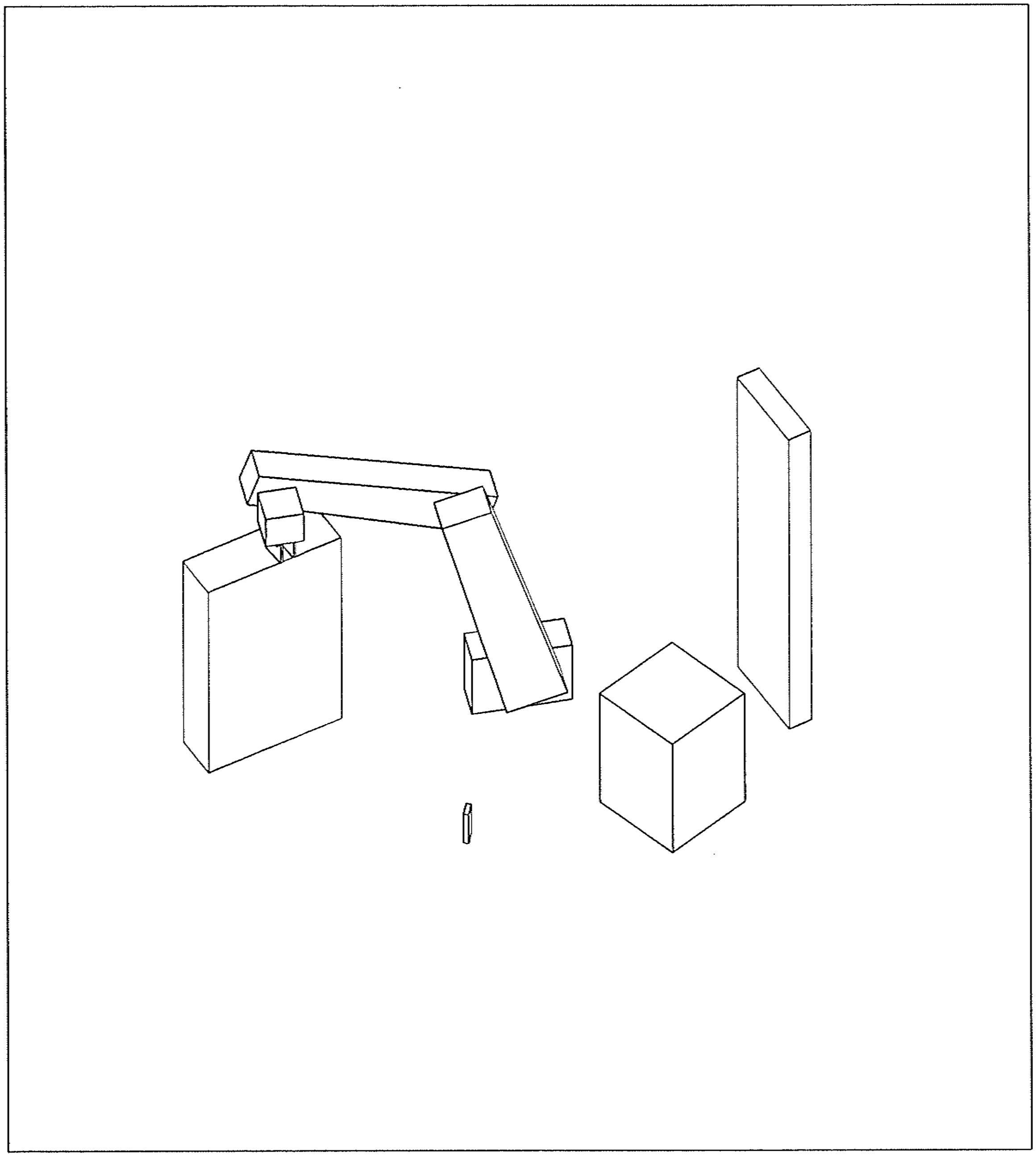
PATH 3 STEP 8
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



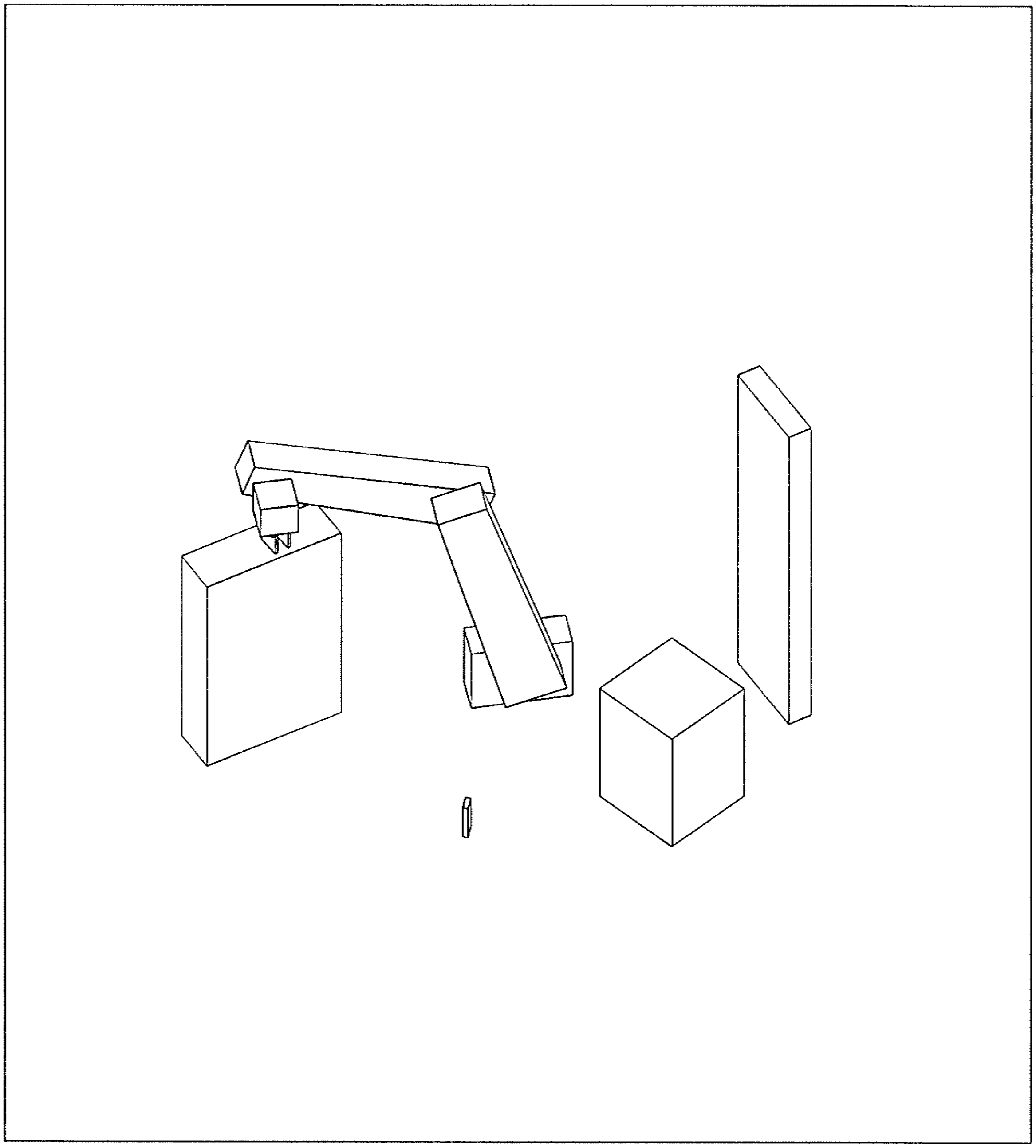
PATH 3 STEP 9
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



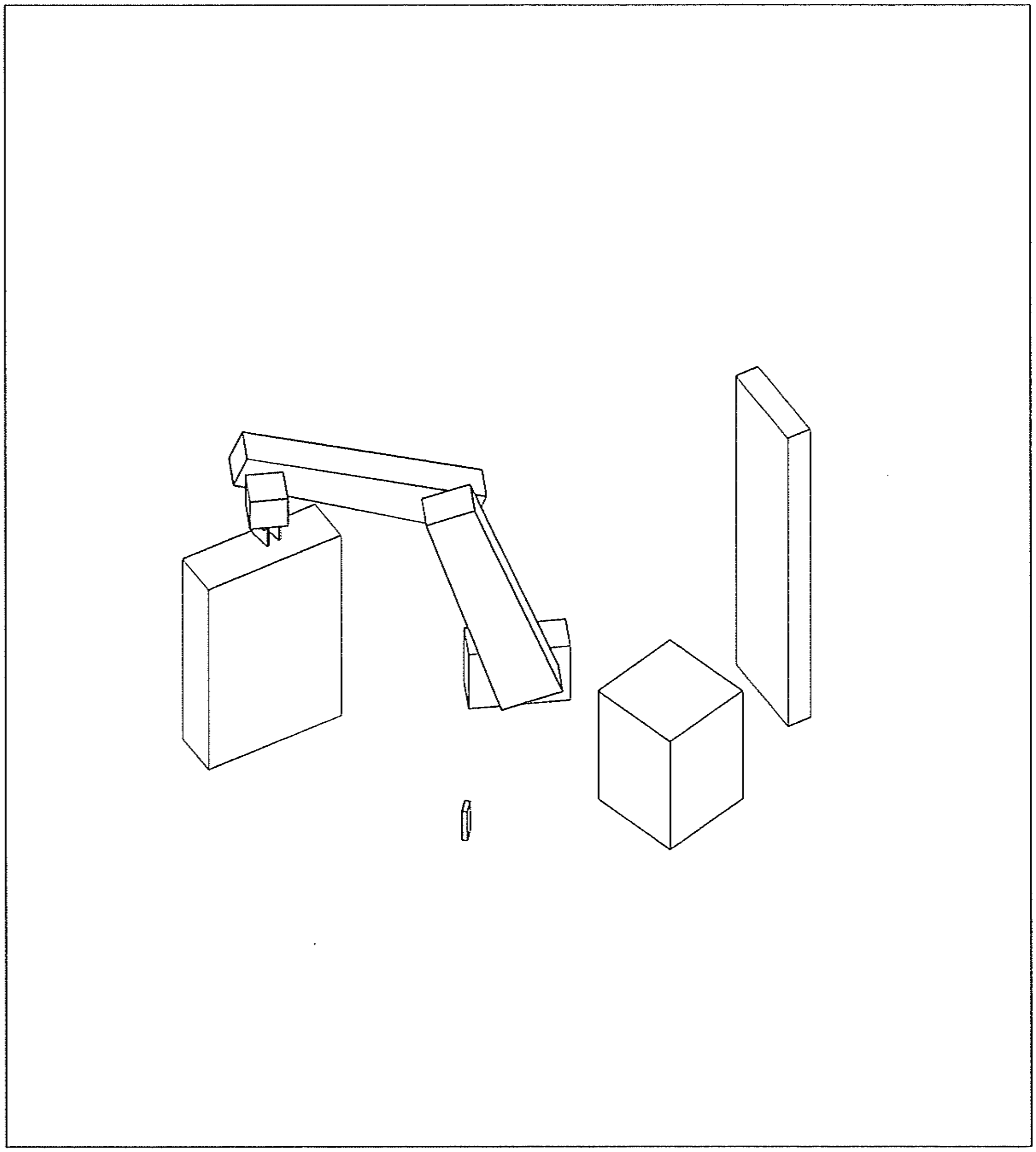
PATH 3 STEP 10
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



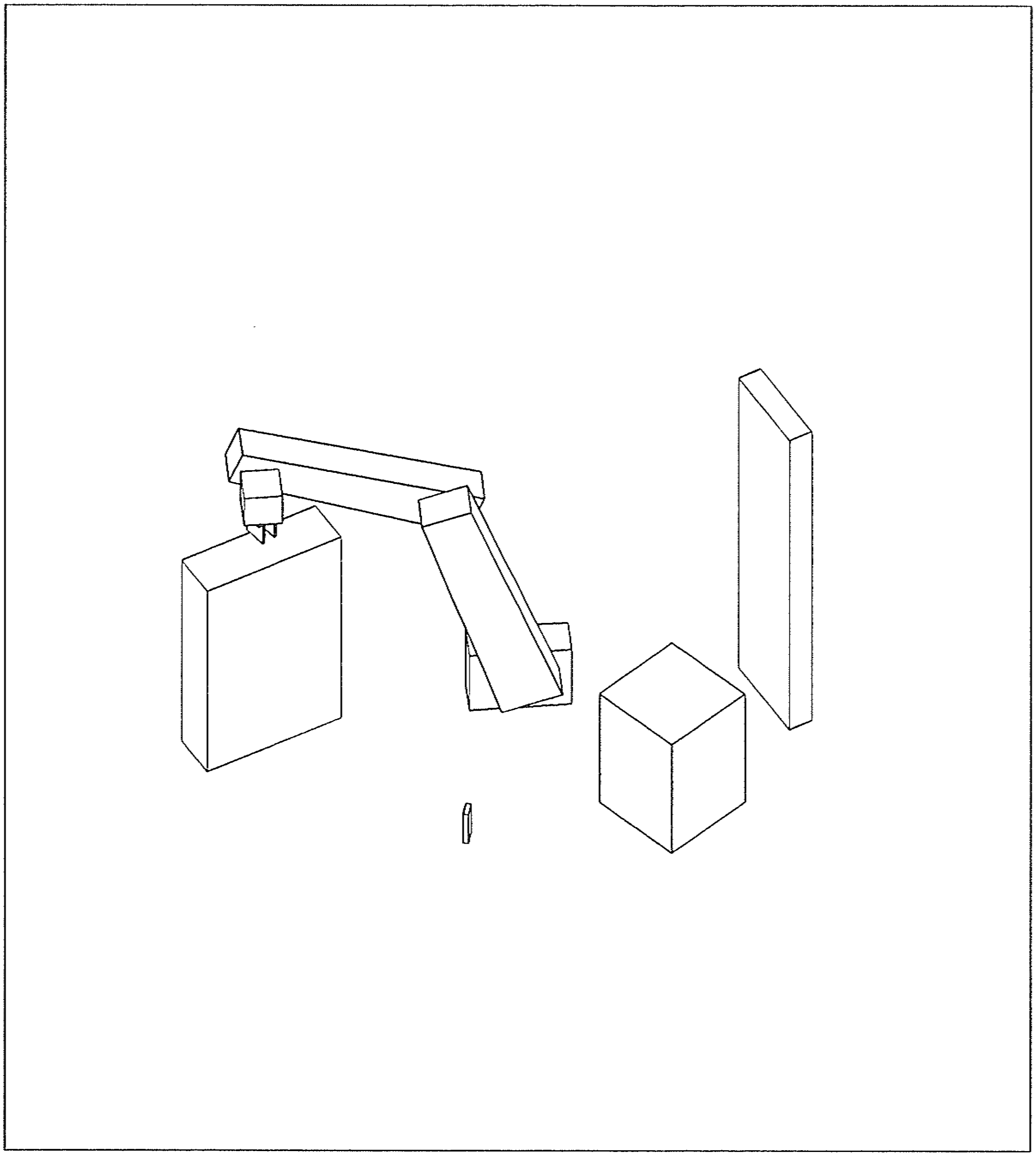
PATH 3 STEP 11
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



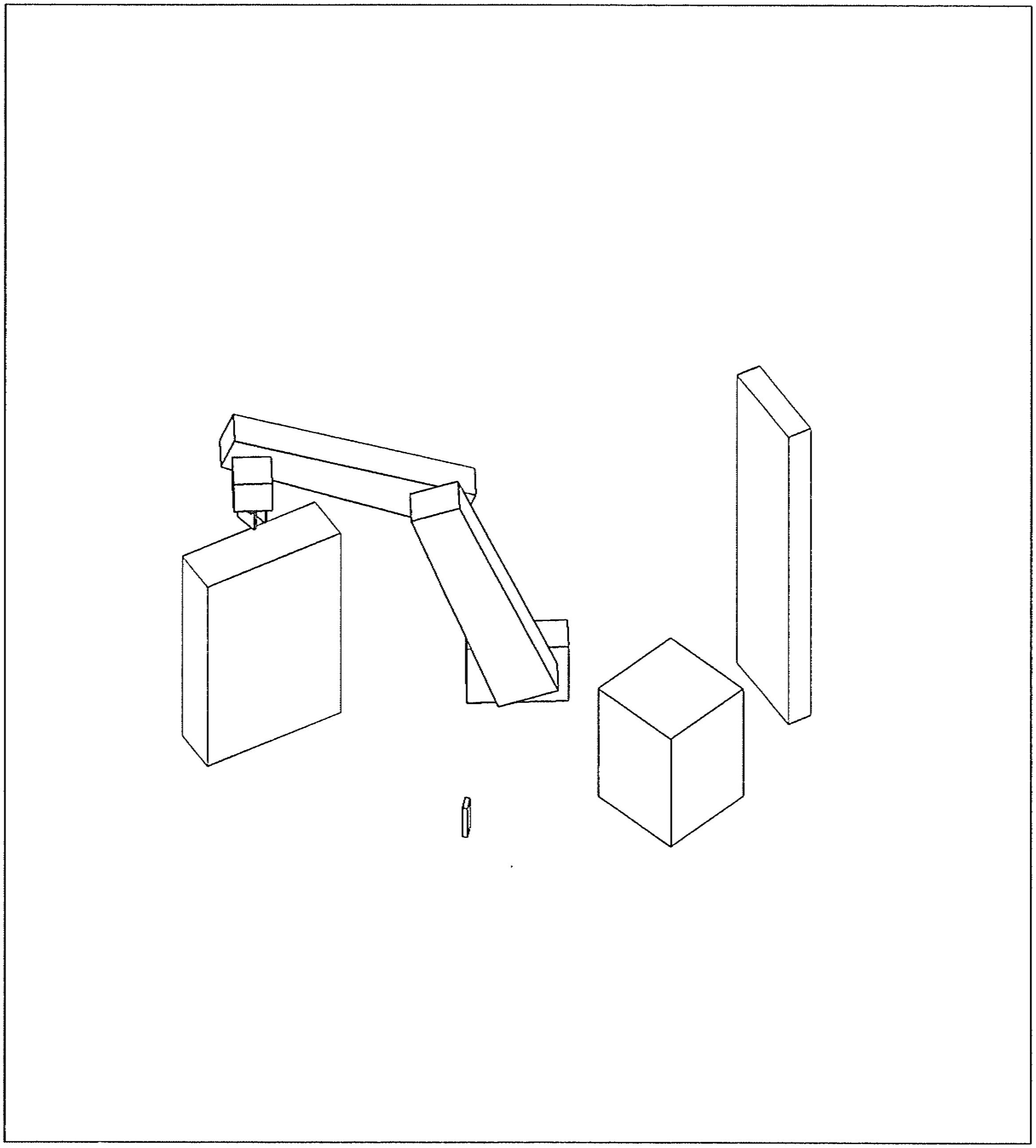
PATH 3 STEP 12
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



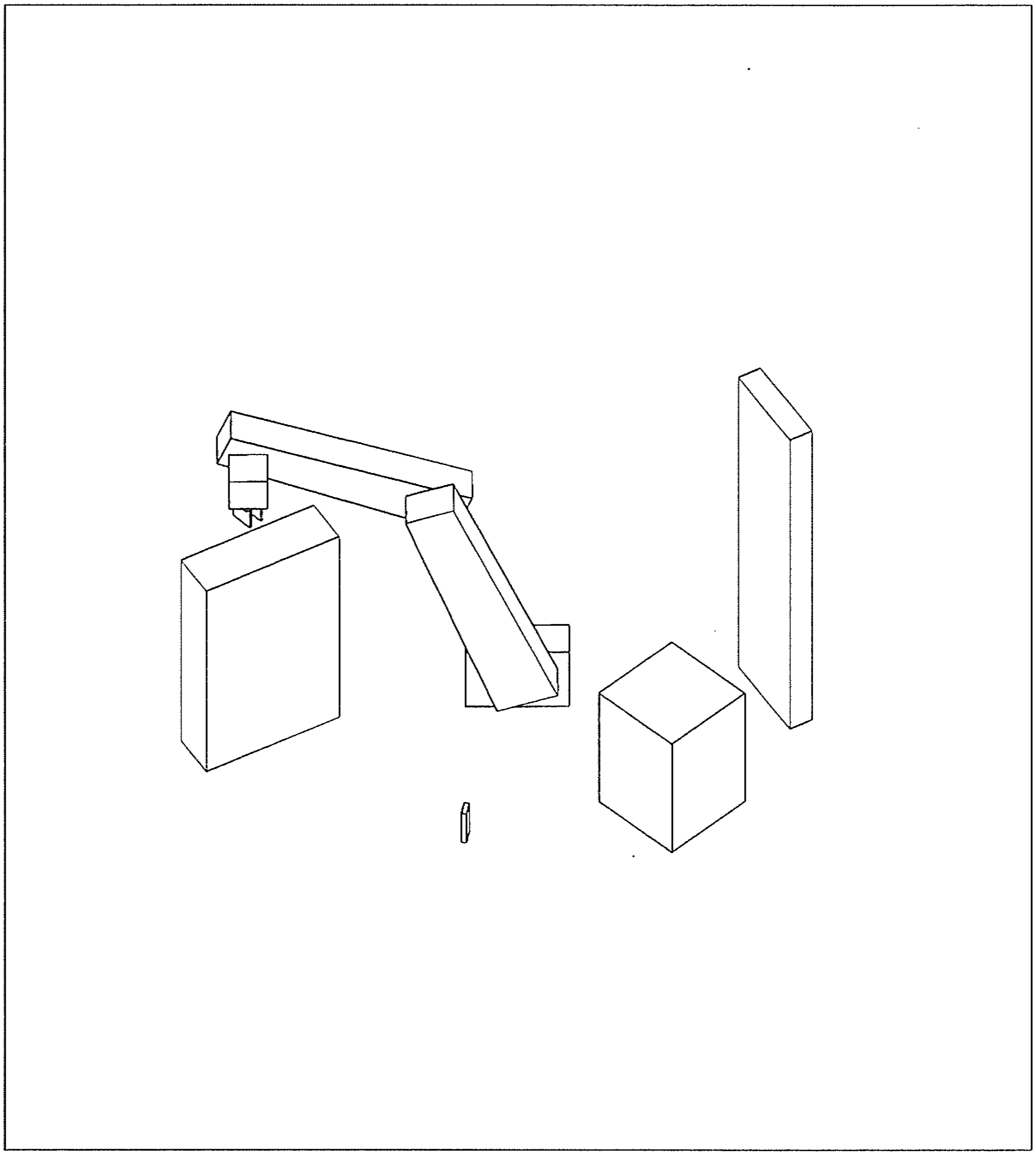
PATH 3 STEP 13
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



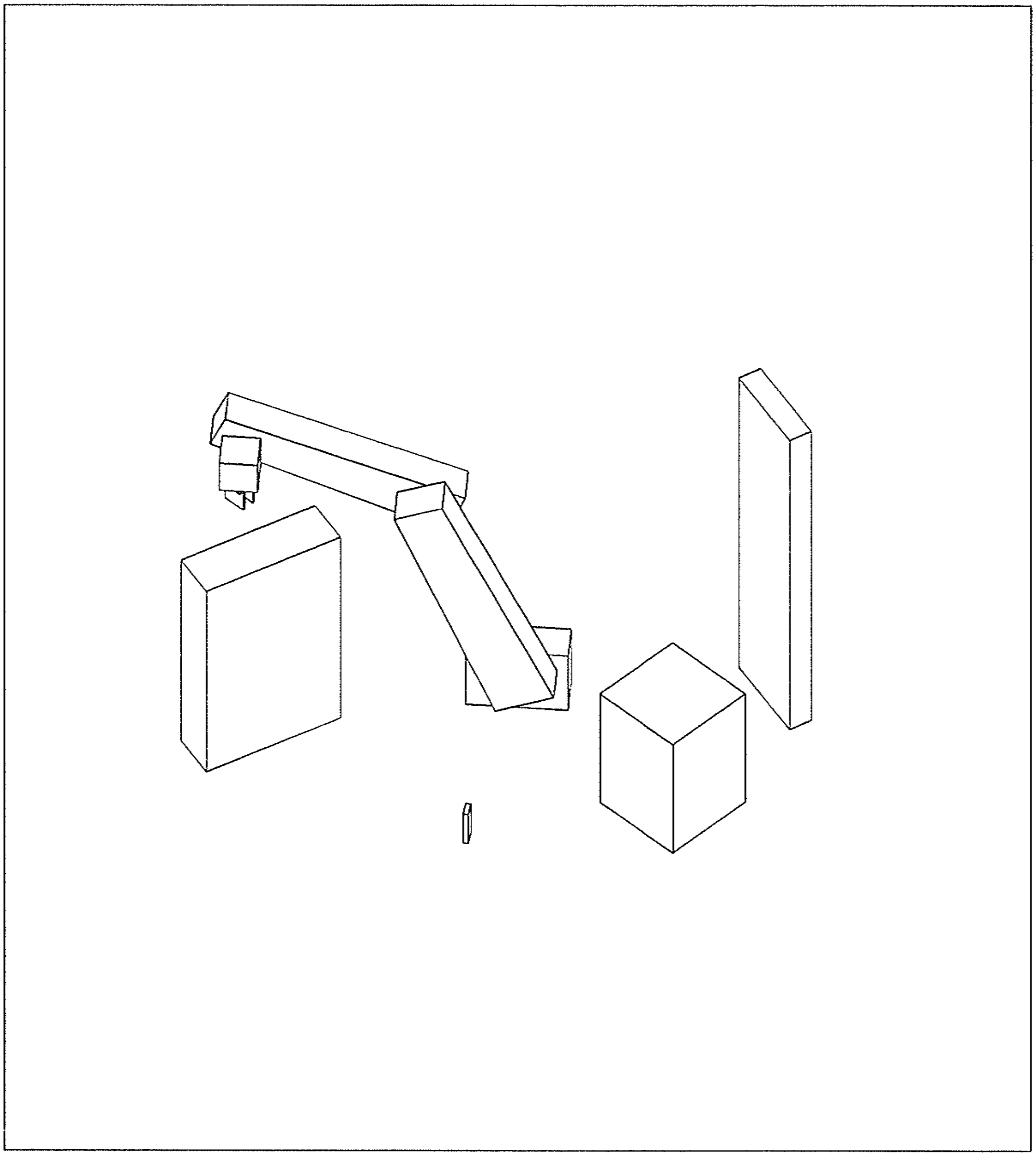
PATH 3 STEP 14
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



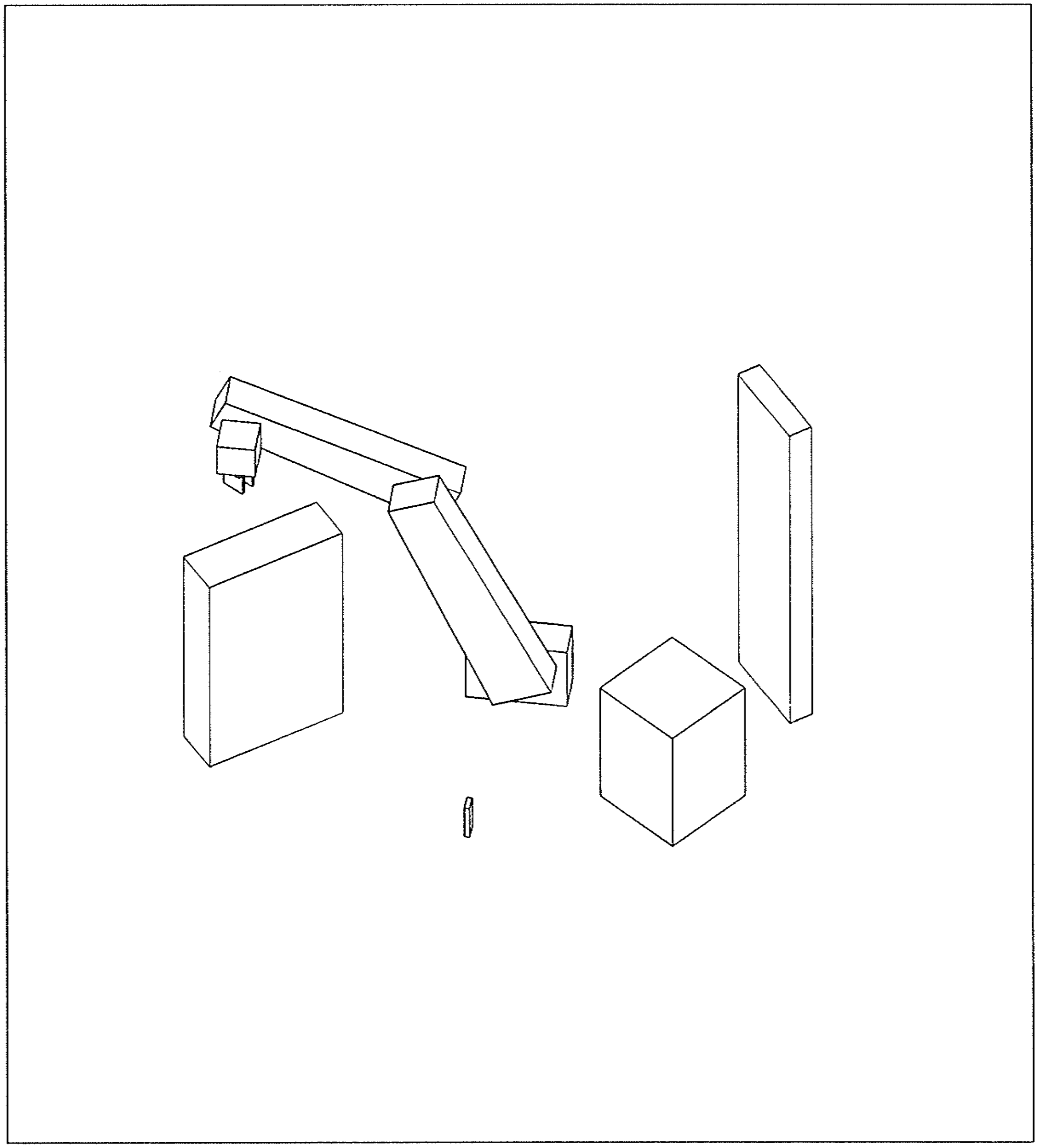
PATH 3 STEP 15
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



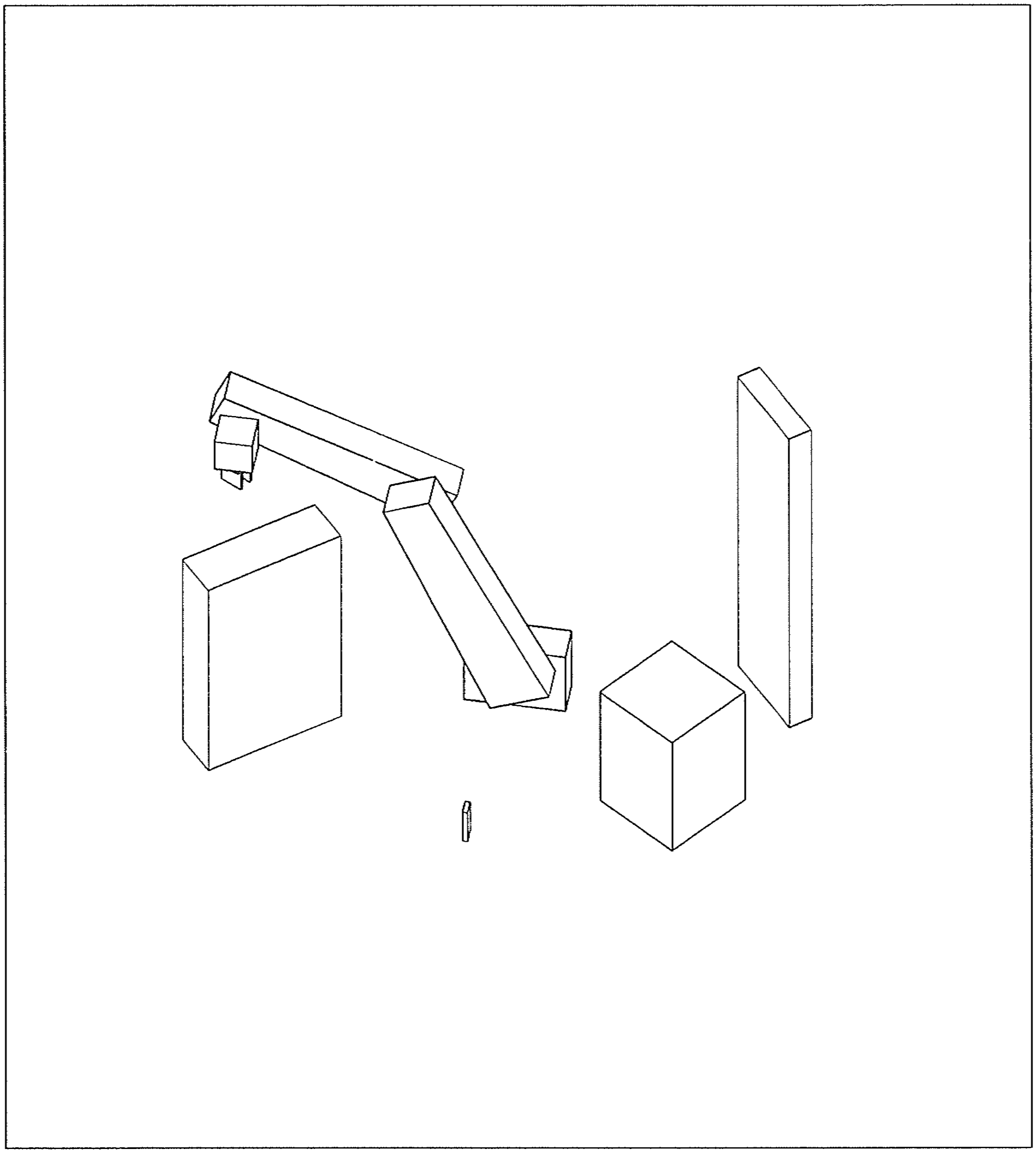
PATH 3 STEP 16
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



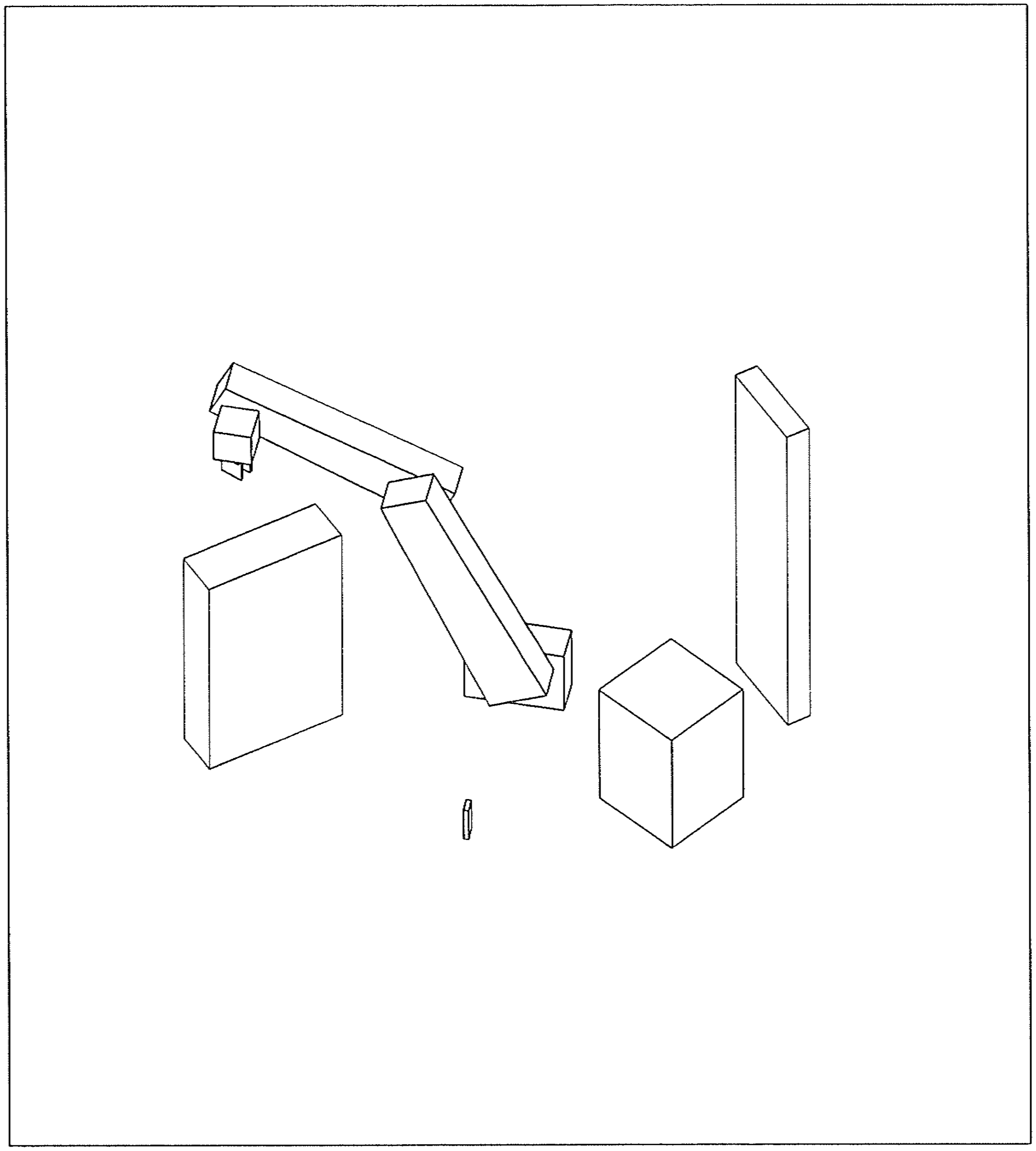
PATH 3 STEP 17
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



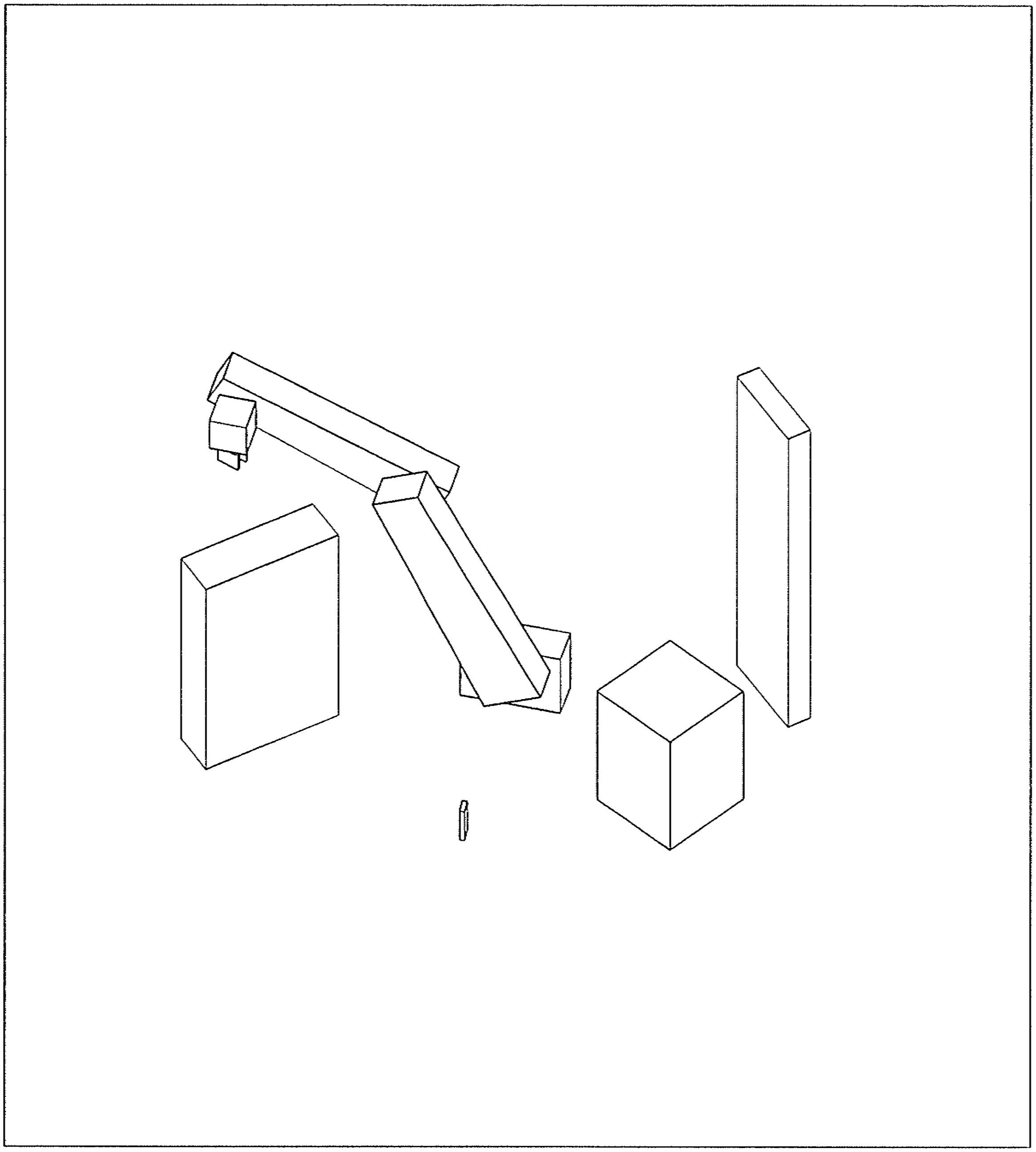
PATH 3 STEP 18
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



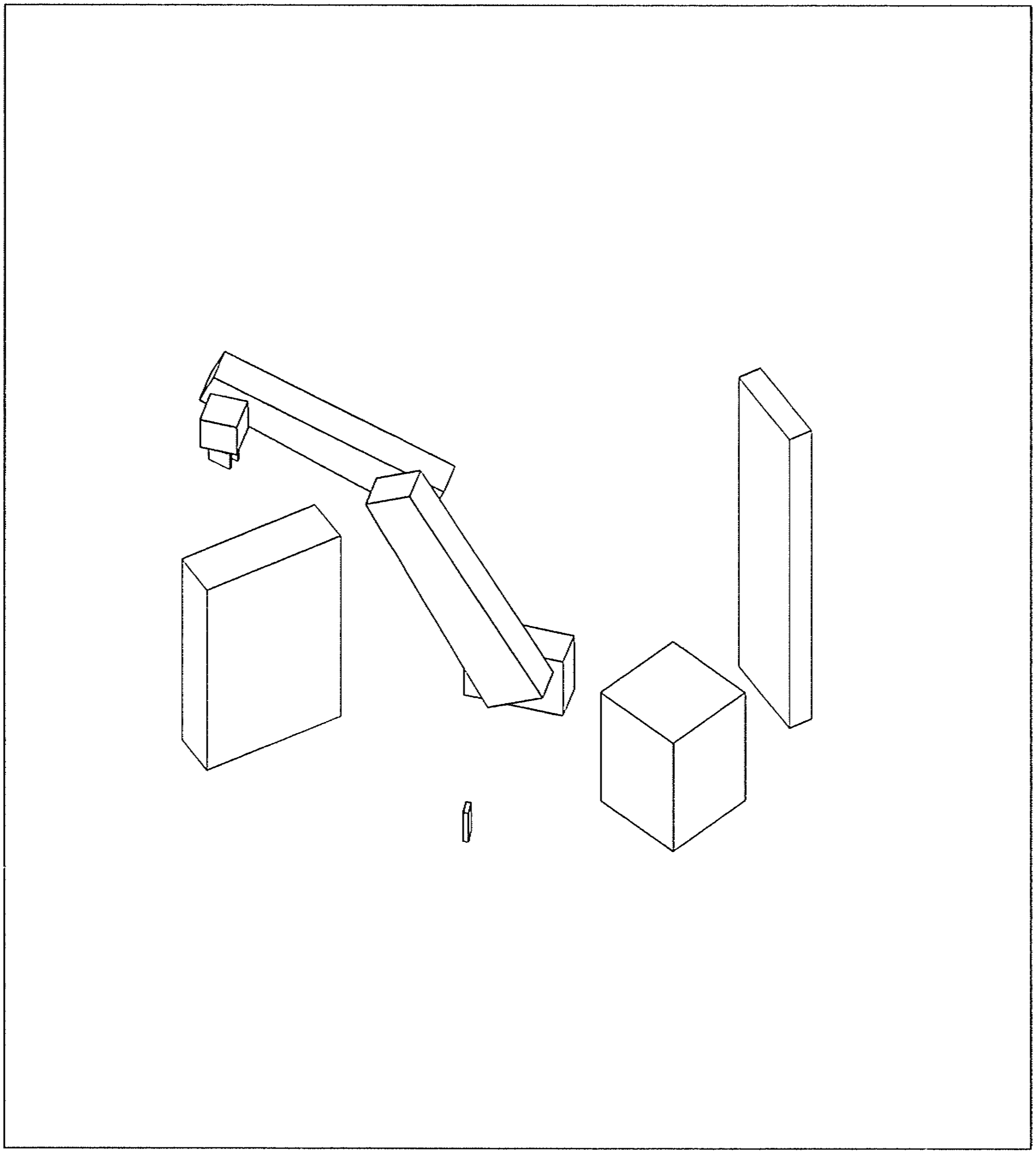
PATH 3 STEP 19
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



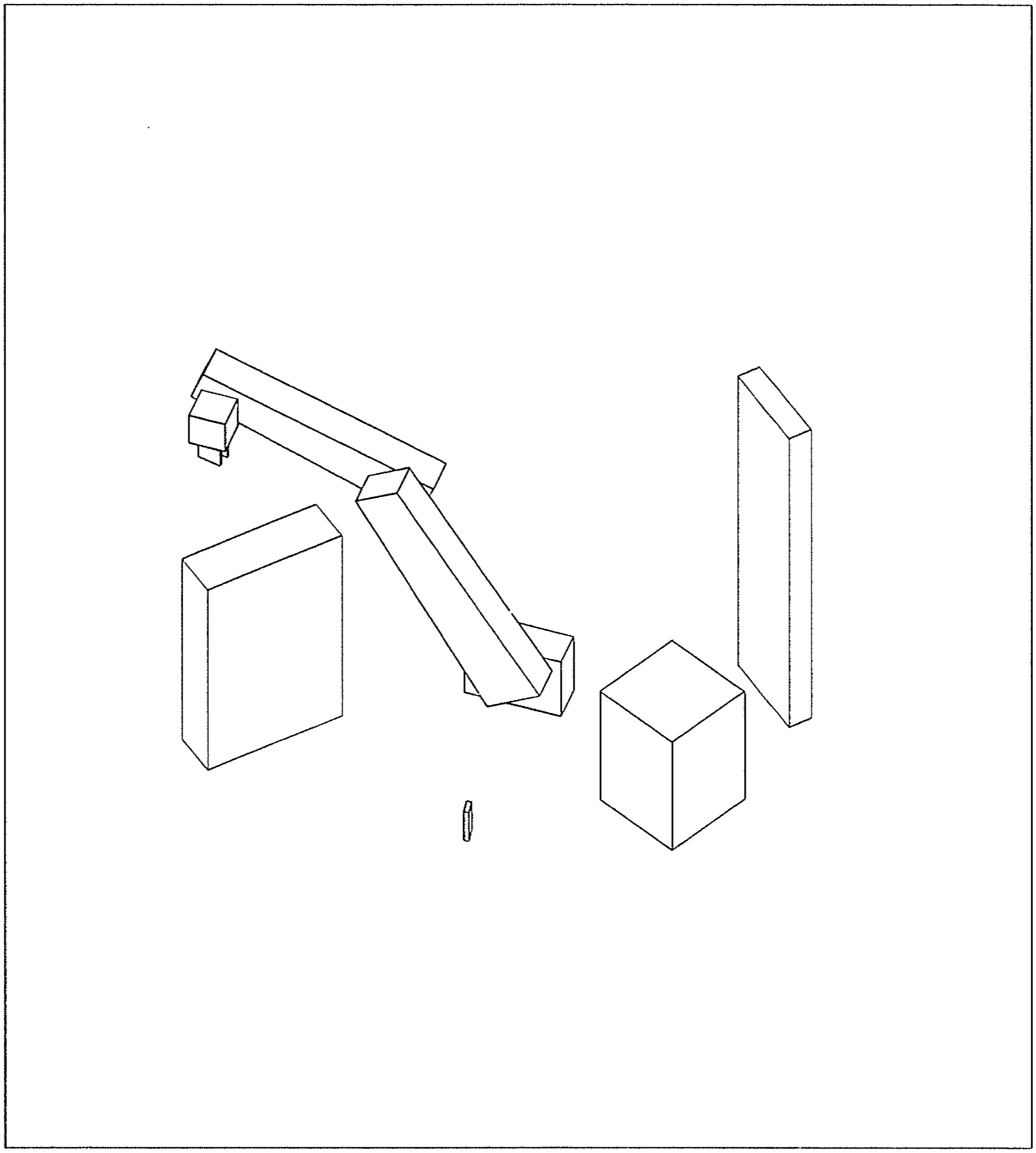
PATH 3 STEP 20
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



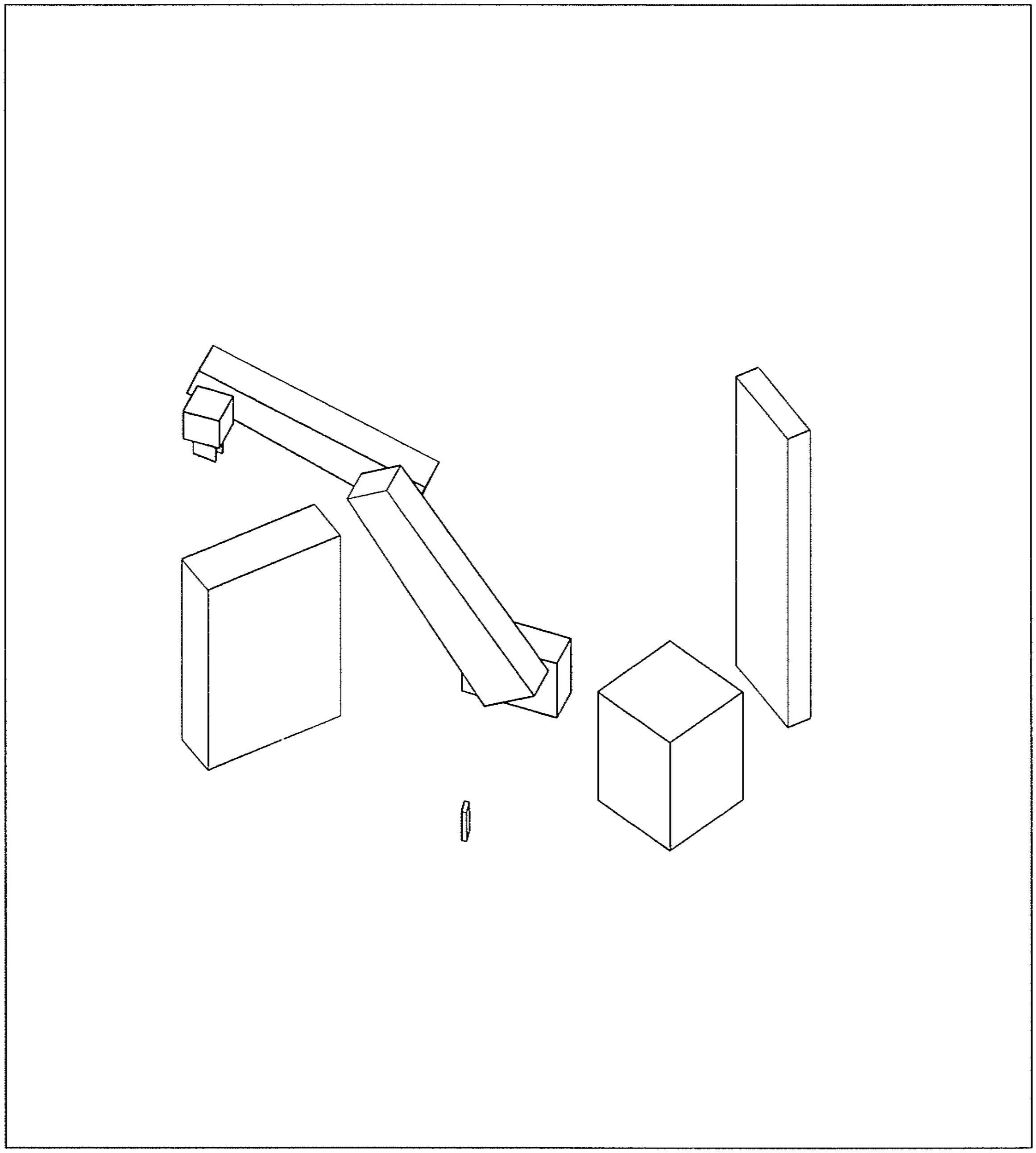
PATH 3 STEP 21
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



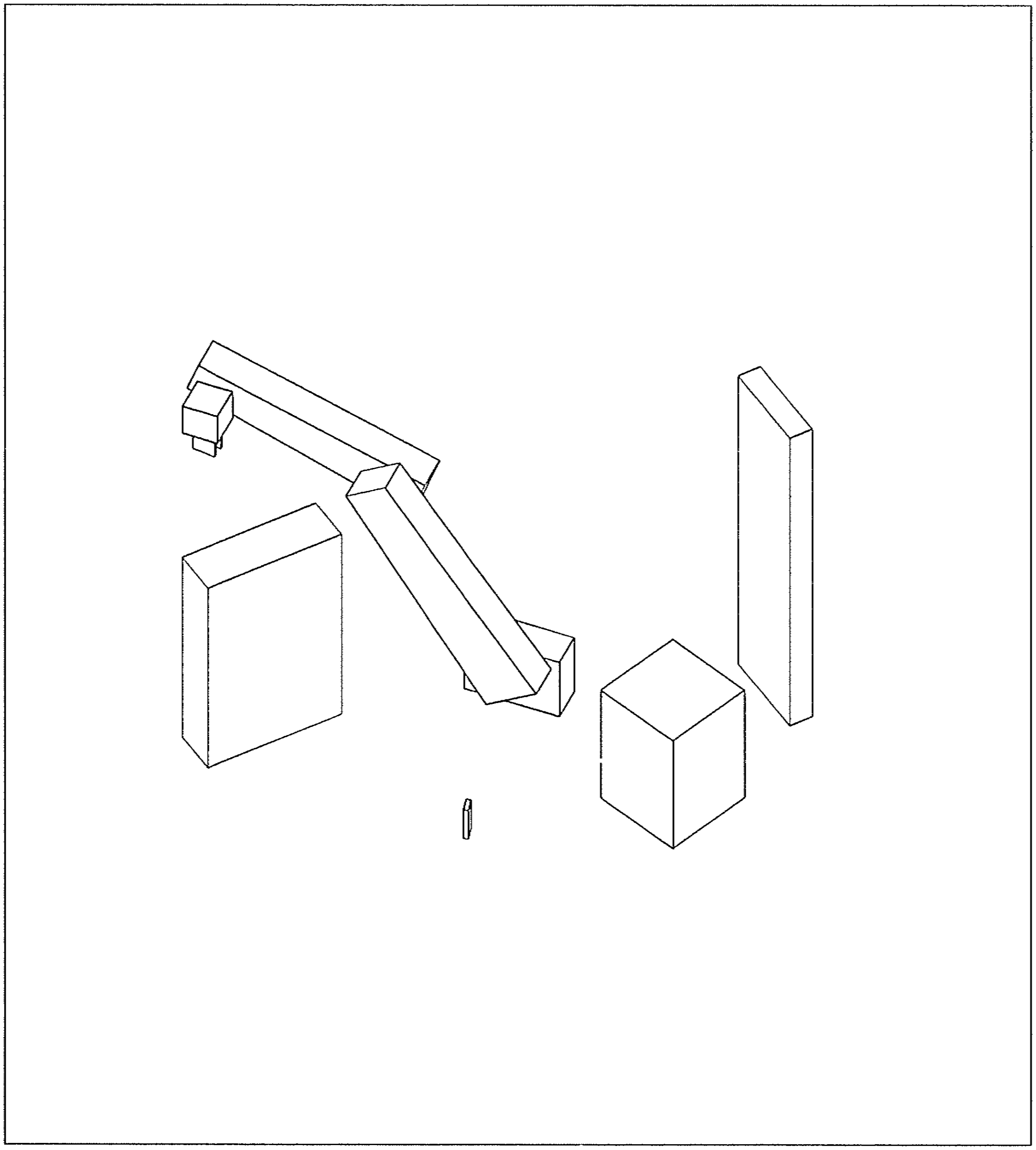
PATH 3 STEP 22
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



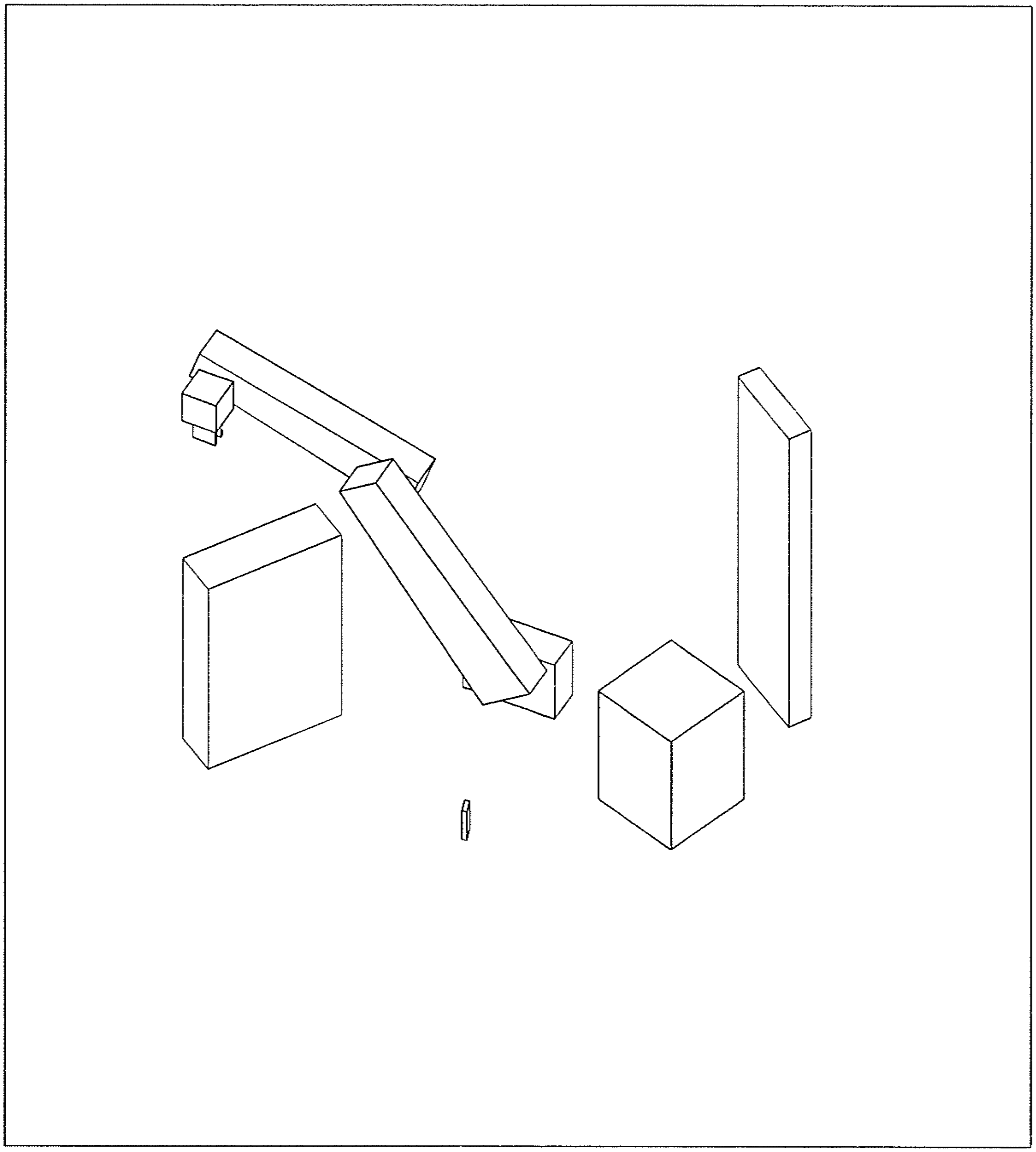
PATH 3 STEP 23
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



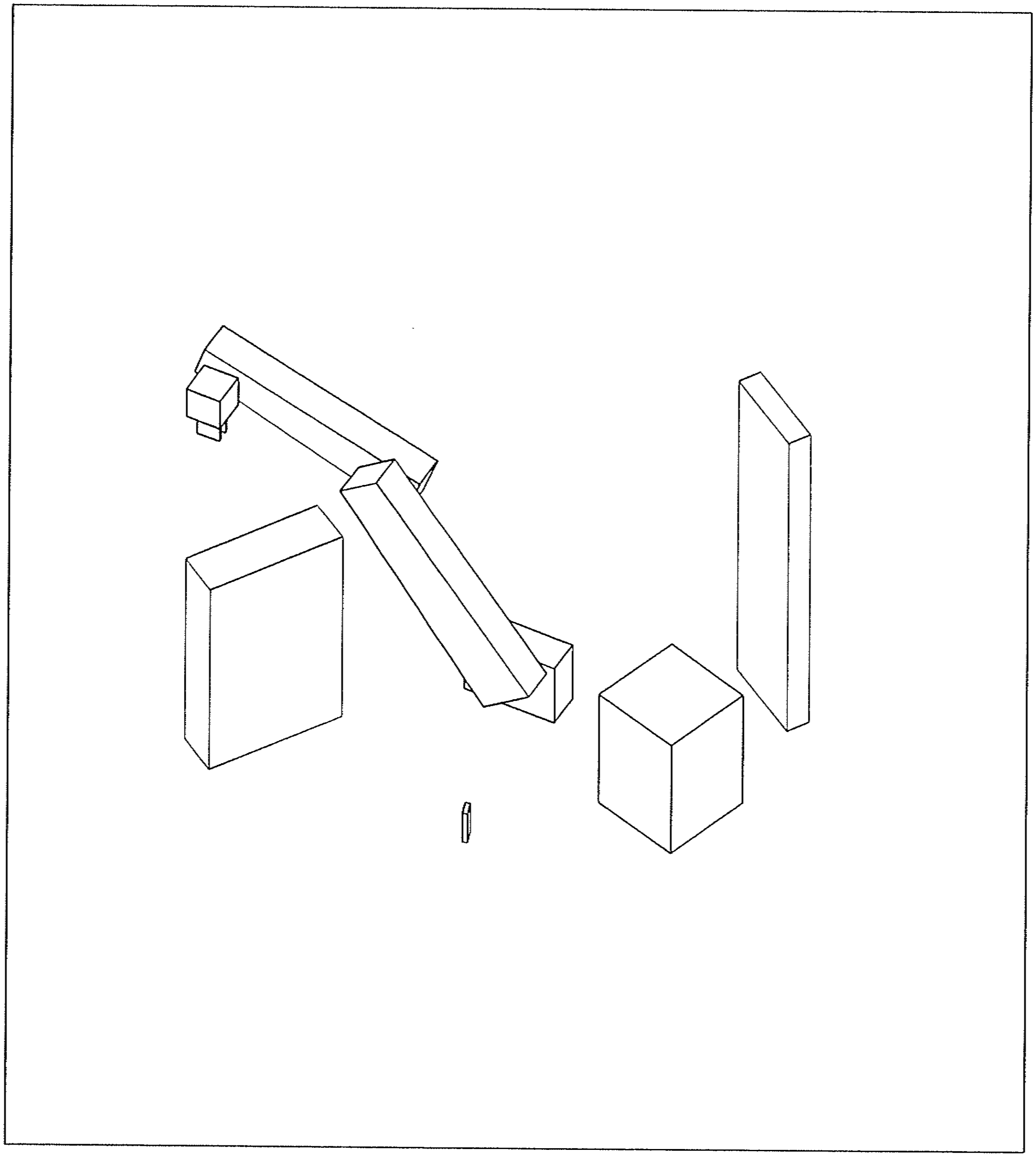
PATH 3 STEP 24
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



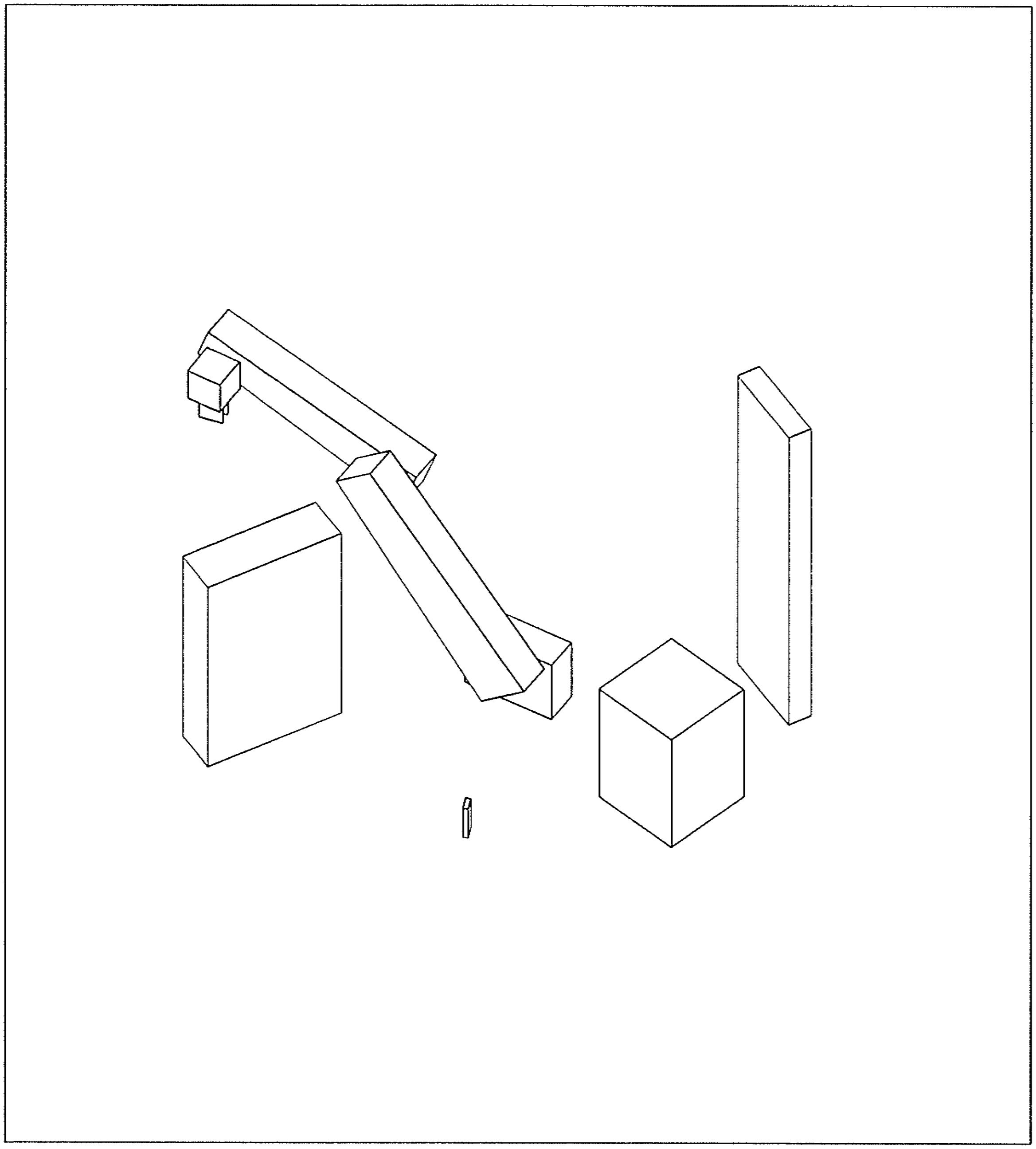
PATH 3 STEP 25
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



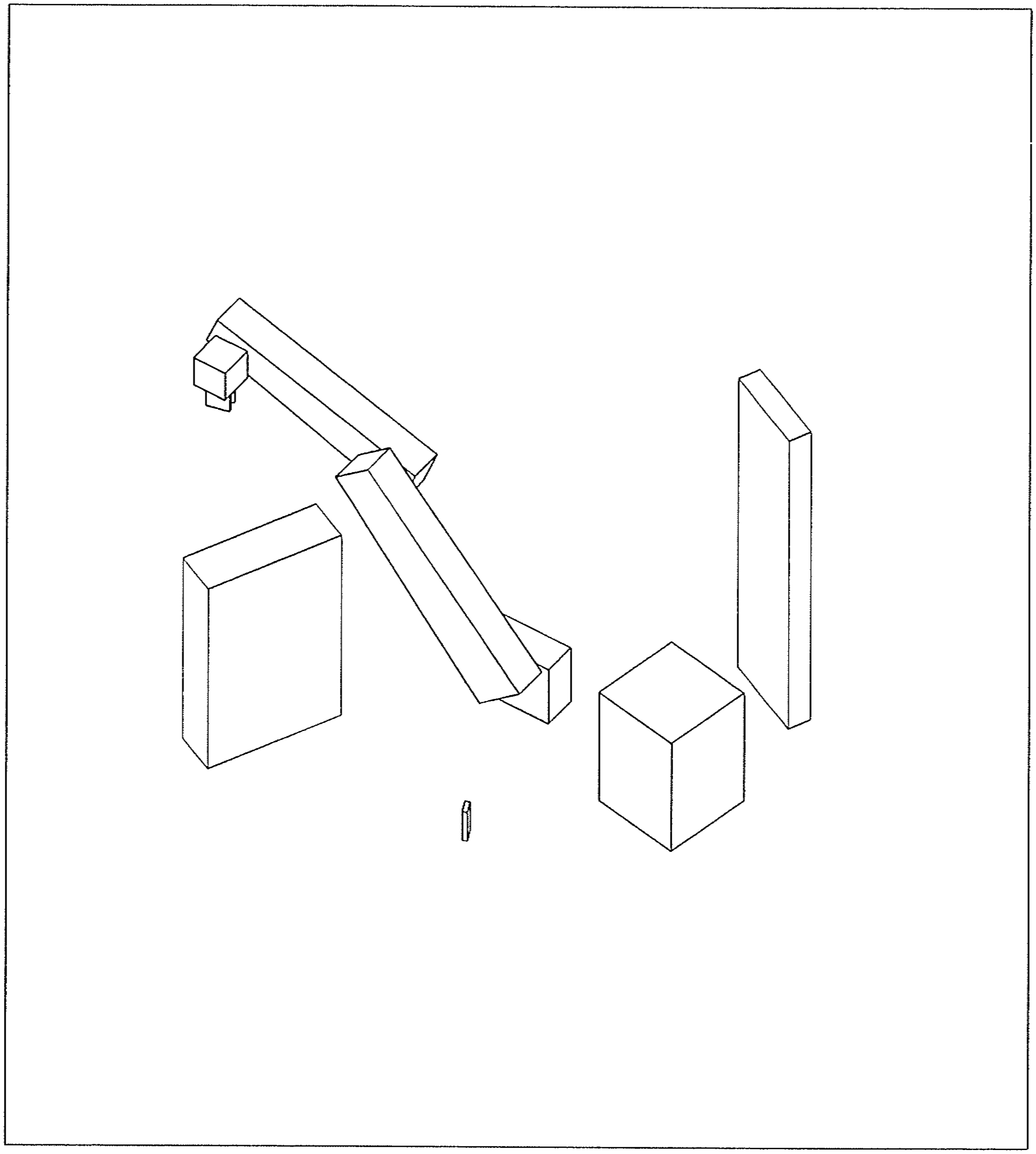
PATH 3 STEP 26
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



PATH 3 STEP 27
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



PATH 3 STEP 28
SAFE VARIABLE CONFIGURATION
PATH IS SAFE



PATH 3 STEP 29
SAFE FIXED CONFIGURATION
PATH IS SAFE

... FINDPATH OUTPUT ENDS