REFERENCES

[1] Carson,G.B., Bolz,H.A., and Young,H.H., *Production Handbook*, 3rd Edition, 1972, pp. 19-39 to 19-45.

[2] Barnes,R.M., *Motion and Time Study - Dcdgn and Measurement* of *Work,* 6th edition, John Wiley and Sons, 1968, pp. 496-510.

[3] Vaughn,R.C., *Introduction to Industrial Engineering,* Iowa State University Press, 1967, pp. 373-384.

[4] J. Nevins, D. Whitney, and S. Drake, D. Killoran, M. Lynch, D. Seltzer, S. Simunovic, R. M. Spencer, P. Watson, and A. Woodin, *Exploratory Research In Industrial Modular Assembly,* Third Progress Report, No. R-921, Charles Stark Draper Laboratory, Inc., Cambridge, Massachusetts, August 1975.

[5] Paul,L., *Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm,* Stanford Artificial Intelligence Project, Memo AIM- 177, March 1973.

[6] Binford,T.O., Grossman,D.D., Miyamoto,E., Finkel,R., Shimano,B.E., Taylor,R.H., Bolles,R.C., Roderick,M.D., Mujtaba,M.S., Gafford,T.A., *Exploratory Study of Computer Integrated Assembly Systems,* Second Progress Report covering the period September 1974 to November 1975, Stanford Artificial Intelligence Laboratory.

[7] Paul,L., *ARM.LOU[UP,DOC],* Internal documentation resident on the system at Stanford Artificial Intelligence Laboratory, October 1974.

APPENDIX 1: MTM STUDY OF ASSEMBLY BY HUMAN BEING

		TMU* =.0006 MIN	JIFFIES =1/60 SEC
REACH TO HANDLE	R 24 A	14.9	32
GRASP HANDLE	G 1 A	2.8	4
MOVE TO FIXTURE	M 18 C	13.5	23
POSITION	P 1 NS	10.4	22
RELEASE	RL 1	2.0	4
REACH TO BASE	R 16 A	12.3	27
GRASP BASE	G 1 A	2.0	4
MOVE TO FIXTURE	M 16 C	18.7	40
POSITION	P 2 NS	21.0	45
RELEASE	RL 1	2.0	4
REACH TO SPINDLE	R 24 A	14.9	32
GRASP SPINDLE	G 1 A	2.0	4
ROVE TO FIXTURE	M 24 C	25.5	55
POSITION	P 2 NS	21.0	45
ROVE 1 INCH	M 1 C	3.4	7
6 TIMES			
TURN 180 DEG	T+AP 180 S 3.4	20.3	
RELEASE	RL 1 2.0	4.3	
UNTURN	T+AP 180 S 9.4	20.3	
GRASP	G 1 A 2.0	4.3	
	6 X 22.8=136.8	43.2=295	
REACH TO SHELL	R 21 A	14.0	30
GRASP	G 1 A	2.0	4
MOVE TO FIXTURE	tl 21 c	23.8	51
POSITION	P 2 NS	21.0	45
TURN AN0 APPLY PRESSURE	T+AP 45 S	3.5	8
RELEASE	RL 1	2.0	4
HOVE BACK	R 30 A	17.5	38
TOTAL		386.2	823
		=0.232 MIN	=13.9 SEC

* TIME MEASUREMENT UNIT

EXPLANATION OF SPECIFIC CASES OF ASSEMBLY PRIMITIVES[3]

REACH R 24 A

R stands for **REACH,** 24 for 24 inches, and **A** one of the following categories:

A Reach to object in fixed location, or to object in other hand or on which other hand rests.

B Reach to single object in location which may vary slightly from cycle to cycle.

c Reach to object jumbled with other objects in a group so that search and select occur.

D Reach to a very small object or where accurate grasp is required.

E Reach to indefinite location to get hand in position for body balance or next motion or out of way.

MOVE M 14 A

M stands for MOVE, **14** for distance of **14** inches, and **A** one of the **following** categories:

A Move object to other hand or against stop.

B Move object to approximate or indefinite location.

c Move object to exact location.

TURN & APPLY PRESSURE T & AP 180 S

T & AP stands for **TURN & APPLY PRESSURE, 180** for a turn of 180 degrees, and **S** one of the following ranges of weight that is turned:

S Small - 0 to **2** lb.

M Medium - 2.1 to 10 lb.

L Large - 10.1 to 35 lb.

POSITION P 1 NS

P stands for **POSITION, 1** for class of fit, **NS** for non-symmetry of the part.

1 Loose fit, no pressure required.

2 Close fit, light pressure required.

3 Exact fit, heavy pressure required.

RELEASE RL 1

RL stands for **RELEASE,** for one of the two cases:

1 Normal release performed by opening fingers as independent motion.

2 Con tact release.

GRASP G 1 A

G stands for **GRASP, 1** for **a** category, **A** for subcategory as shown.

1 Pick up grasp.

1 A Small, medium or large object by itself, easily grasped.

1 B Very small object or object lying close against a flat surface.

1C Interference with grasp on bottom and one side of nearly cylindrical object of following subclasses.

1C1 Diameter larger than **1/2** inch.

1C2 Diameter **1/4** inch to **1/2** inch.

1C3 Diameter less than **1/4** inch.

2 Regrasp.

3 Transfer Grasp.

4 Object jumbled with other objects **so search** and select occur,

4 A Object larger than 1 * 1 * 1 inch'.

4 B Object **1/4 * 1/4 * 1/4** to **1 * 1 * 1 inch** ^{3}.

4 C Object smaller than **1/4 * 1/4 * 1/8 inch** ^{3}.

6 Contact, sliding or **hook** grasp.

APPENDIX 2: MTM STUDY OF ASSEMBLY BY MACHINE

This analysis of the movements of the Yellow Arm is based on estimates of the times taken to make various motions, as programmed under WAVE. Each movement is allowed **a** grace period of 20 jiffies (60 jiffies-1 sec). The time for each joint to complete its motion is computed as the distance or angle it has to move multiplied by the time taken to move per unit distance or angle, based on a desired maximum average velocity. Estimated time for each motion is the maximum time over all six joints.

The aim was to record and measure the movements that looked reasonable in an attempt to compare the actual assembly time with the estimated time for a working assembly. No attempt was made to try to optimize the assembly time, or to run the arm at a higher speed.

Two different analyses are made, one to compare t-he movements of the arm with human movements, the second to make use of motion primitives that were used in the analysis of the washer gearbox by Draper Lab.[4] In doing the second, analysis, it was assumed that the only tool used by the arm was the hand consisting of the two fingers with binary touch sensors, and that this tool was not replaced.

		JIFFIES =1/60 sec	DRAPER MOTION PRIMITIVES
PUT HANDLE			
OPEN	0	50	Release
GOTO HA_GR	tl	200	Position
POSITION	P	50	*P. Position
CENTER	C	50	Grasp
GOTO MAIN FIXTURE	M	140	Position
POSITION	P	140	P. Position
OPEN HAND (DROP HANDLE)	0	30	Release
WAIT	W	50	*Wait
CLOSE	C	20	Grasp
PUSH INTO POSITION	P	50	Accommodate (Depress)
LIFTOFF	M	30	Position
	Subtotal	810	
PUT BASE			
OPEN	0	50	Release
GOTO BA_GR (APPROACH)	M	140	Position
POSITION	P	50	P. Position
CENTER	C	60	Grasp
GOTO APPROACH OF MAIN FIXTURE	M	140	Position
PLACE IN POSITION	P	140	P. Position

OPEN HAND	0	30	Release
MOVE UP	M	140	Position
CLOSE HAND	C	50	Grasp
MOVE DOWN	P	140	Accommodate (Depress)
LIFTOFF	M	30	Position
	Subtotal	970	

PUT SPINDLE

GET SPINDLE

OPEN AND	0		Release
GOTO APPROACH OF SP GRIP	M	140	Position
POSITION	P	45	P. Position
CLOSE HAND	C	140	Grasp
	Subtotal	325	

PLACE-SP I NDLE

GOTO TOP OF MAIN FIXTURE	M	380	Position
TOUCH TOP OF FI XTURE	P	110	Interface
MOVE OUT	M	80	Position
GOTO SIDE OF MAIN FIXTURE	M	380	Position
TOUCH SIDE	P	140	Interface
MOVE OUT	M	50	Position
SEARCH FOR HOLE (ASSUME GET RIGHT FIRST TIME)			
GOTO TOP OF HOLE	M	200	Position
MOVE DOWN	P	80	Accommodate (I nser t)
TWIST ANO FORCE DOWN	T+AP	80	Accommodate (Complex accommodate)
OPEN HAND	0	45	Release
MOVE UP A BIT	M	35	Position
CLOSE HAND	C	65	Grasp
	Subtotal	1645	

TURN SPINDLE

TURN 120 DEG CLOCKWISE	T+AP	80	Rotate
OPEN HAND	0	30	Release
3 * TURN 120 CCW	3*T+AP	150	Rotate
CLOSE HANO	C	30	Grasp
3 * TURN 120 CW	3*T+AP	240	Rotate
OPEN HAND	0	30	Release
3 * TURN 120 CCW	3*T+AP	150	Rotate
CLOSE HAND	C	30	Grasp

'3 * TURN 120 CW	3*T+AP	240	Rotate
OPEN HAND	0	30	Release
RN 120 CCW	3*T+AP	150	Rotate
CLOSE HAND	C	30	Grasp
2 * TURN 128 CW	2*T+AP	160	Rotate
	Subtotal	*1350*	
ASSEMBLE SHELL			
OPEN HAND AND	0		Release
GOTO APPROACH OF SHELL	M	*380*	Position
POSITION	P	*35*	P. Position
CLOSE HANO (GRASP)	C	*150*	Grasp
, GOTO APPROACH OF BASE	M	*200*	Position
WAIT	w	50	Wait
PUSH DOWN ANO WOBBLE	P	*80*	Accommodate (Complex accommodate)
WAIT	W	*50*	Wait
RELEASE	0	*20*	Release
CLOSE HARD	C	80	Grasp
FORCE DOWN AND WOBBLE	P	*35*	Accommodate (Complex accommodate) **
FORCE DOWN AND TURN 45 DEG CW	T+AP	*80*	Accommodate (Complex accommodate)
	Subtotal	*1160*	
PARK ARM			
OPEN	*0*	40	Release
MOVE UP	M	*50*	)
PARK	M	120	)Return
	Subtotal	*210*	

TOTAL ASSEMBLY TIME 6470 jiffies = 108 sec

*Wait and P.Position (Precise position) are two primitives introduced here that are not used in Draper report.

**This movement is not the peg in hole insertion problem in the true sense: rather it is the insertion of a round hole over an oval peg with a large clearance until there is an interfacing.

VI. MATHEMATICAL TOOLS FOR VERIFICATION VISION

Robert C. Balks

Artificial Intelligence Laboratory
Computer Science Department
Stanford University

The author is *a* graduate student in the Computer Science Department.

CHAPTER 1

INTRODUCTION

Verification Vision (VV), as defined in [BOLLES 75], has three main concerns:

(a) the confidence that the system is finding the correct object(s),

(b) the precision within which the system has located the object(s),

and (c) the cost involved in determining this information.

For each task, such as visually locating a rivet hole, the assembly engineer specifies the desired confidence and precision, and possibly some cost limits such as the maximum real time that the task can take. During the execution of the task, the VV system gathers more and more information until the confidence and precision requirements have been met or until some cost limit has been exceeded.

The VV system that will be discussed in this paper gathers information by applying "operators," such as edge operators, correlation operators, and region growers, which are designed to locate and describe "features," such as line segments, correlation points, and regions. The information produced by such operators can be roughly classified into two types: value information and position information. Value information includes such things as the value of a correlation coefficient, the contrast across an edge, and the intensity of a region. Position information, in addition to (x,y) or (x,y,z) information, may include orientation information. For example, an edge operator can return the (x,y) position of a point on a line and an estimate of the orientation of the line. The same edge operator may return the contrast across the edge and the confidence that there really is an edge at that place, both of which would be classified as value information.

The distinction between value information and position information is made because often it is reasonable to assume that the values from different operators are independent, but it is seldom reasonable to assume that the positions of features are independent (especially features of rigid objects). "Independence" means that knowing the value of one operator (such as a correlator) does not affect the expected value for another operator (such as an edge

operator). The position information, on the other hand, is not independent because the location of one point or the orientation of one line greatly influences the possible positions for other features.

Figure 1.1.1 shows the general flow of control for a VV system based upon these ideas. The flowchart suggests several important questions which this or any similar VV system has to be able to answer:

(1) Given a specific set of objects, what are some candidate features and what operators can be used to find such features?

(2) What information can a specific operator contribute toward increasing the confidence that the correct object is being found?

(3) What is the expected cost of applying operator X?

(4) What was the actual cost of applying operator X?

(5) Which operator should be applied next?

(6) How can the results of several operators be combined to give an overall confidence?

(7) How can the results of several operators be combined to determine an estimate for the location of the object and a precision about that estimate?

(n) What is the expected number of operators required to achieve a certain confidence?

These questions can be partitioned according to the time at which they are most important. For example, the question about the expected number of operators is important at "planning time" when the system or user is trying to decide the expected cost of accomplishing the task. The question about candidate features is important at "programming time" when the user is describing potential sources of information. This paper divides a VV task into four times, or stages:

(1) PROGRAMMING TIME: the user states the goal of the task, gives the confidences, precisions, and costs for the task, and interactively chooses potential features and operators.

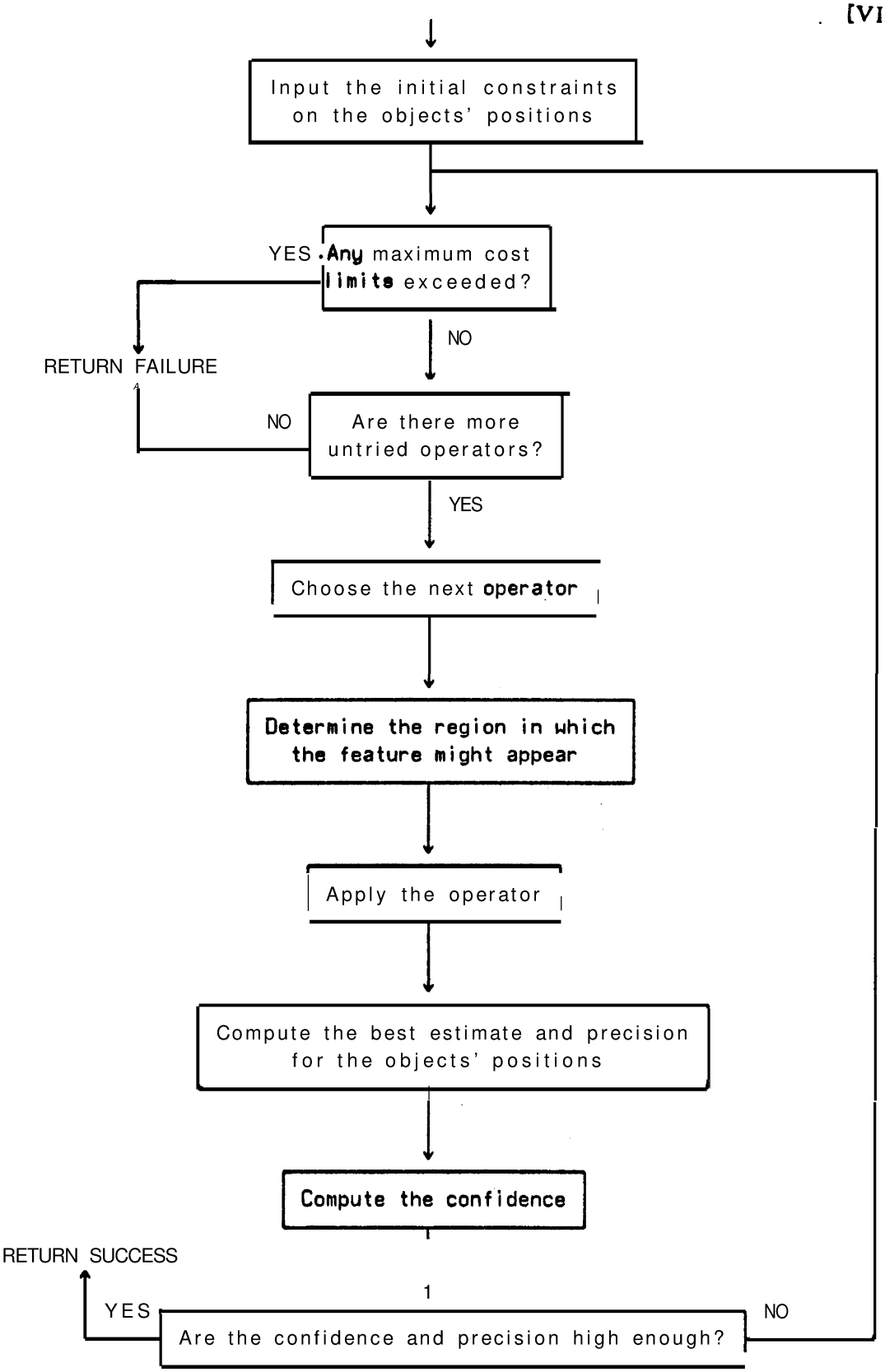

Input the initial constraints on the objects' positions
YES
Any maximum cost limits exceeded?
NO
RETURN FAILURE
NO
Are there more untried operators?
YES
Choose the next operator
Determine the region in which the feature might appear
Apply the operator
Compute the best estimate and precision for the objects' positions
Compute the confidence
RETURN SUCCESS
YES
Are the confidence and precision high enough?
NO

FIGURE 1.1.1

(2) TRAINING TIME: the system applies the operators to several sample pictures and gathers statistical information about the effectiveness of the operators.

(3) PLANNING, TIME: the system ranks the operators according to their expected contribution, determines the expected number of operators to be needed, and predicts the cost of accomplishing the task.

(4) EXECUTION TIME: the system applies operators one at a time, combines the results into confidences and precision, and stops when the desired levels have been reached or until a cost limit has been exceeded.

This paper concentrates on the mathematics required at the execution and planning times. It describes methods for answering the questions about the contributions of operators and how to combine the results of several operators. It is less concerned about how the features and operators are suggested initially. The basic approach is to use a least squares technique to combine the available information to form a current, best estimate for the location of the object (plus a tolerance about that estimate) and Bayesian probability within a sequential pattern recognition scheme to compute the necessary confidences. These are all well-known techniques, but they combine particularly nicely to 'answer the various questions raked within a VV system.

This paper relies heavily upon the domain of programmable assembly for its examples and motivation. The techniques are discussed in the context of a highly controlled en vi ronment in which mechanical arms are performing assembly tasks. Some of the techniques have been optimized to take advantage of specific properties of this environment, but the basic methods used to produce location and confidence information from the results or' several visual operators are more widely applicable. Other promising tasks areas that require similar types of visual information processing are the interpretation of aerial photograph, calibration, and medical diagnosis.

CHAPTER2

EXECUTION-TIMEMATHEMATICS FOR INSPECTION

The description of the relevant mathematics has been separated into two segments, execution-time mathematics and planning-time mathematics. The former is concerned with combining the *actual* results of features as they are found. The later is concerned with computing and combining the *expected* contributions of the features.

The mathematical tools are incrementally developed in conjunction with a sequence of examples that has been designed to incorporate an ordered set of complexities.

Section 1
OPERATOR VALUE INFORMATION

Consider the task of deciding whether or not there is a screw on the end of the screwdriver. For simplicity assume that normalized cross-correlation is the only type of operator known to the VV system. Correlation uses patches from a 'planning' picture as features to be found in the actual (ie. the execution time) picture. Figure 4.1.1 shows a planning picture with the screw on the end of the screwdriver and several sample pictures, some with the screw present, some with it missing. Figure 4.12 shows several correlation "features" outlined on top of the planning picture. When operator 1 is applied to a sample picture it locates a match with a certain value for the correlation coefficient. Correlation coefficient values range from -1 to +1. Figure 4.1.3 shows the results of applying operator 1 to ten different sample pictures of the screwdriver with the screw on the end. If the frequency of these correlation values is assumed to follow a normal distribution, the corresponding distribution can be approximated from the experimental mean and standard deviation of these values. The fitted, sample distribution is shown in figure 4.1.4.

If operator 1 is applied to a sample picture in which the screw is missing, there will not

[VI.6]

PLANTIC PLATES

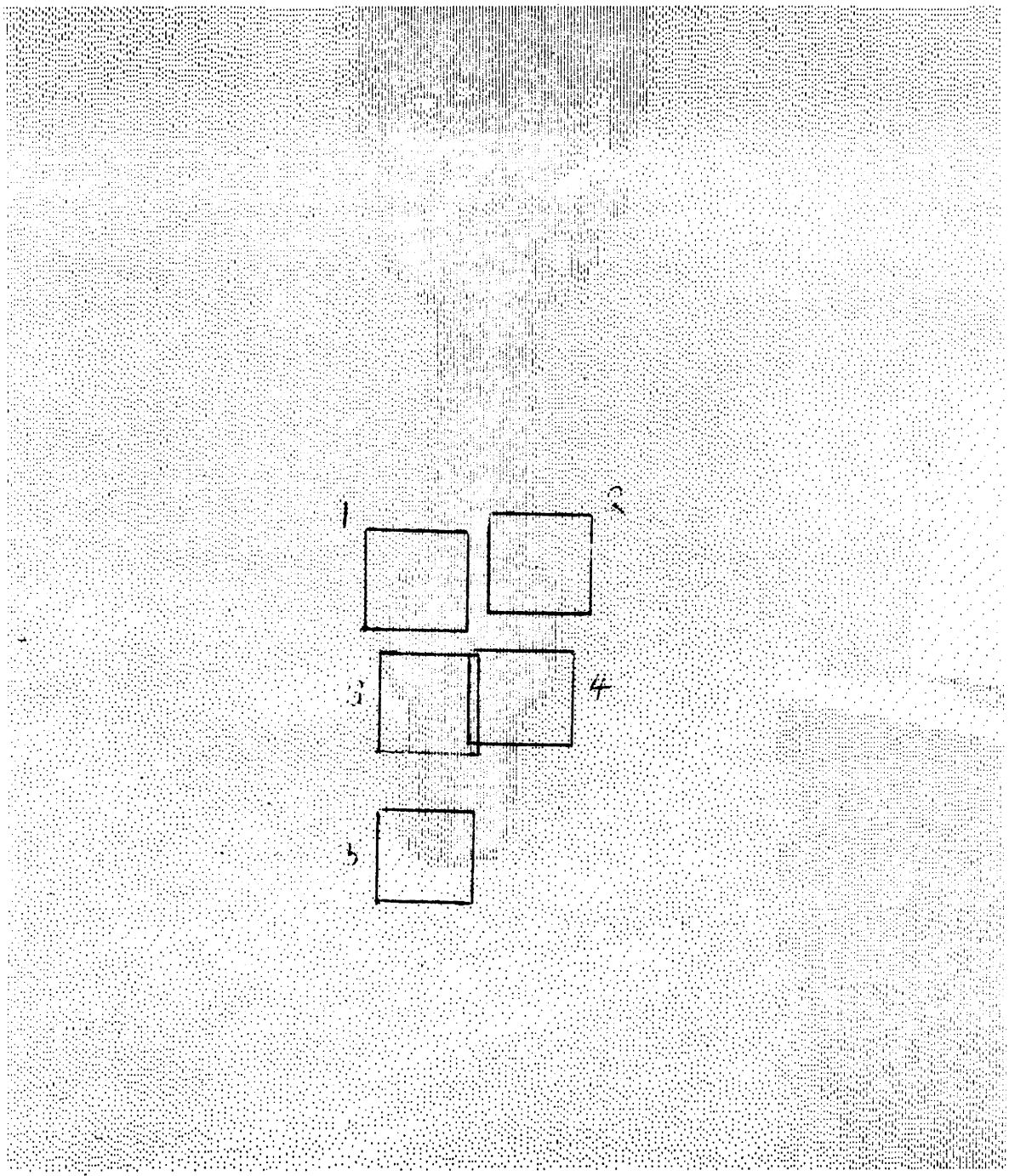


FIGURE 4.1.2

[V1.8]

.80
.88
.77
.85
.83
.89
.96
.86
.85
.82

sample mean .85

sample standard deviation .05

FIGURE 4.1.3

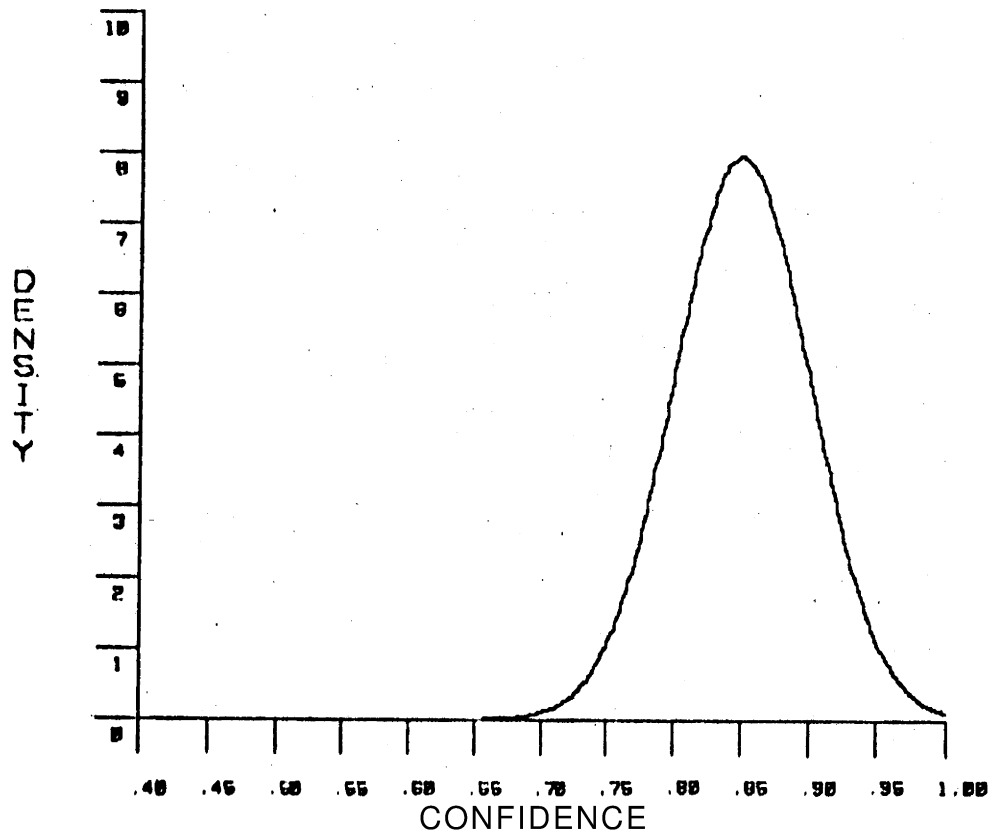


FIGURE 4.1.4

be a portion of the picture that "looks like" operator 1, and hence there will not be a portion of the picture that correlates well with operator 1. Operator 1 will still locate a "best match" but the correlation coefficient will be lower. Thus, operator 1, if reliable, will (1) match the correct piece of the screw, if the screw is there, and (2) match some other feature (with a lower correlation value) when the screw is not there. This performance difference is the basis for deciding whether the screw is there or not.

If operator 1 is applied to several pictures without the screw, the resulting correlation values will form some distribution. A table of ten such trials and the corresponding distribution (again assuming a normal distribution) are shown in figure 4.1.5. The two frequency functions are superimposed in figure 4.1.6.

If operator 1 is applied to a picture for which it is not known whether the screw is there or not, the operator will find a "best match" with some correlation value, eg. .93. Based solely upon operator 1, should the system say that- the screw is there or not? One would probably say that the screw is there, but what is the confidence of that decision? In probabilistic terms, one is interested in the probability that the screw is there, given that operator 1 has a value of .93, ie.

(4.1.1) $$P[\langle \text{screw there} \rangle \mid \langle \text{value(operator 1)=.93} \rangle].$$

Let

(4.1.2) $$H \equiv \langle \text{the screw is on the end of the screwdriver} \rangle$$
$$v1 \equiv \langle \text{value(operator 1)} = X \rangle$$

then Bayes' theorem (eg. see [Hoel 71]) expresses the desired *a posteriori* probability in terms of the *a priori* and conditional probabilities as follows:

(4.1.3) $$P[H|v1] = \frac{P[v1|H] * P[H]}{P[v1|H] * P[H] + P[v1|\neg H] * P[\neg H]}$$

or

(4.1.4) $$P[H|v1] = \frac{1}{1 + \dfrac{P[v1|\neg H] * P[\neg H]}{P[v1|H] * P[H]}}.$$

These formulas state the desired probability in terms of probabilities that are often more

.60
.71
.50
.57
.67
.70
.61
.76
.65
.71

sample mean .65

sample standard deviation .07

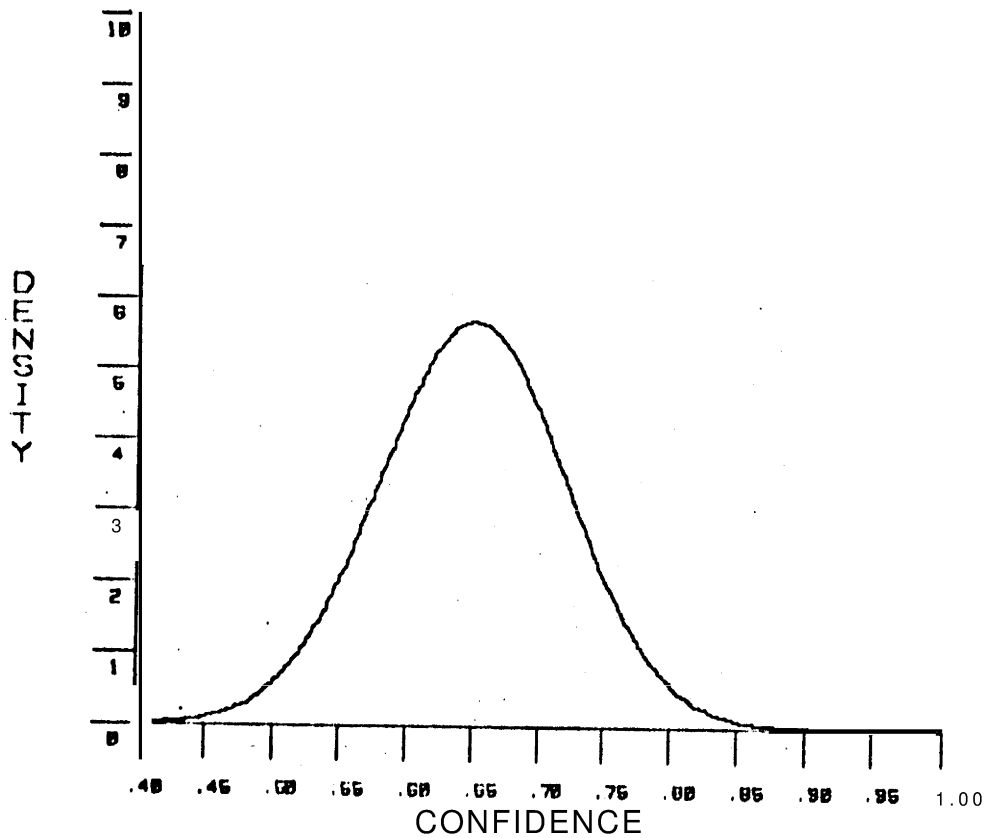


FIGURE 4.1.5

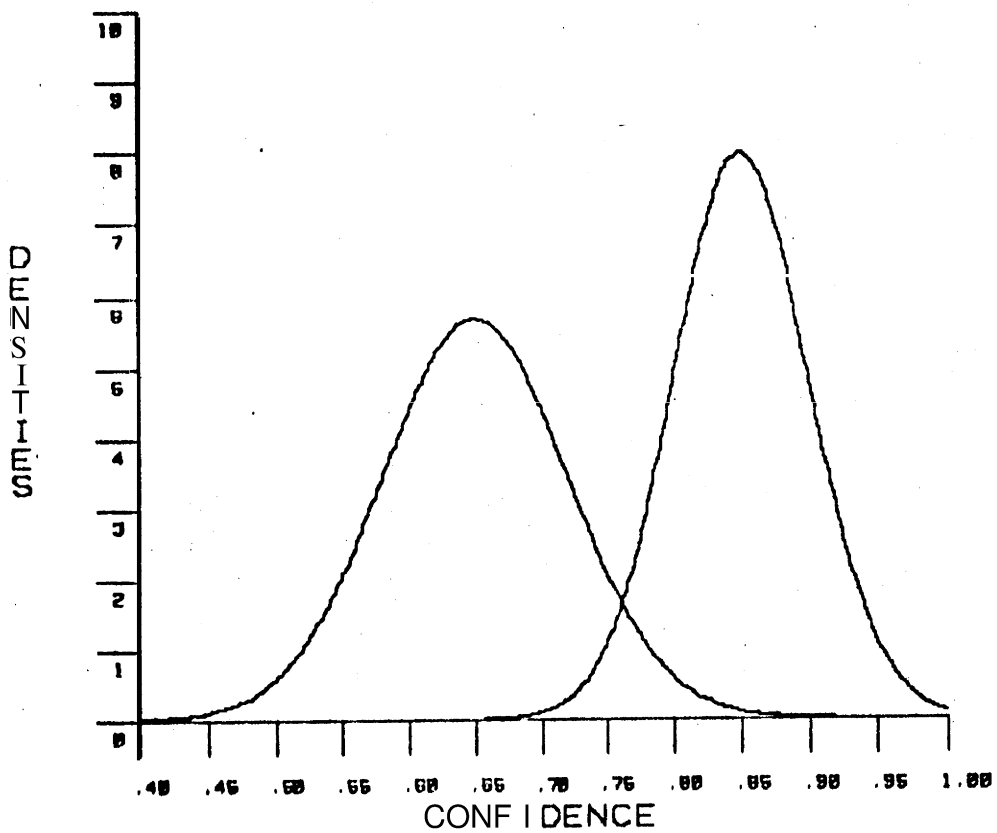

DENSITIES
10
9
8
7
6
5
4
3
2
1
0
.40
.45
.50
.55
.60
.65
.70
.75
.80
.85
.90
.95
1.00
CONFIDENCE

FIGURE 4.1.6

readily computed. The *a priori* probabilities are based upon measured statistics or the experience of the assembly engineer. For example, if the screwdriver correctly acquires a screw nine-tenths of the time, P[H] should be set to .9. The density functions shown in figure 4.1.6 can be used to compute the conditional probabilities, P[v1|¬H] and P[v1|H]. Since the functions are density functions, the probability of the operator producing any one particular value is zero. But the probability of the operator producing a value within a certain range is the integral of the function over that range. Thus one way of estimating the above ratio for a specific value of the operator is to consider a small range about the value, compute the two probabilities by integration, and form the ratio. Notice, however, that as the width of the region decreases, the approximations for the ratio approach the ratio of the two values of the density functions at X. That is, the ratio of the probabilities can be replaced by the ratio of the densities. This observation makes is particularly easy to compute the appropriate ratio fot any value of the operator.

Eayes' theorem can be extended to combine the values of several operators:

$$(4.1.5) \qquad P[H|v1,v2,\ldots vN] = \frac{1}{1 + \dfrac{P[v1,v2,\ldots vN|\neg H]}{P[v1,v2,\ldots vN|H]} * \dfrac{P[\neg H]}{P[H]}}.$$

Since

$$(4.1.G) \qquad P[v1|H] \equiv \frac{P[v1,H]}{P[H]}$$

and

$$(4.1.7) \qquad P[v1,v2|H] = \frac{P[v1,v2,H]}{P[H]} * \frac{P[H,v2]}{P[H,v2]} = P[v1|H,v2] * P[v2|H],$$

then, more generally, the conditional probabilities can be expanded into:

$$(4.1.8) \qquad P[v1,v2,\ldots vN|H] = P[v1|H,v2,v3,\ldots vN] * P[v2|H,v3,v4,\ldots vN] *$$

$$\ldots$$

$$* P[v(N-1)|H,vN] * P[vN|H].$$

If the v's are assumed to be conditionally independent, ie.

(4.1.9) $$P[vj|H,vj+1,\ldots vN] = P[vj|H]$$

then these probabilities reduce to

(4.1.10) $$P[v1,v2,\ldots vN|H] = P[v1|H] * P[v2|H] * \ldots * P[vN|H],$$

and Bayes' theorem becomes

(4.1.11) $$P[H|v1,v2,\ldots vN] = \frac{1}{1 + \prod_{i=1}^{N} \frac{P[vi|\neg H]}{P[vi|H]} * \frac{P[\neg H]}{P[H]}}.$$

In this form it is apparent that the *contribution* of an operator is the value of the ratio:

(4.1.12) $$\frac{P[vi|\neg H]}{P[vi|H]}.$$

The contribution of an operator is the amount of influence that the operator's value has on the estimate of the overall probability of H. The inverse of ratio 4.1.12, ie.

(4.1.13) $$\frac{P[vi|H]}{P[vi|\neg H]},$$

is known as the likelihood ratio. The logarithm of the likelihood ratio is also important, as the chapter on planning-time mathematics will show. The larger the likelihood ratio, the stronger the evidence that the screw is present. This formulation agrees with one's intuition in several ways. Consider figure 4.1.7 in which three values of the operator have been indicated: W, X, and Y. If the operator happens to produce the value W, the likelihood ratio is 1.0, and the probability of the screw being there is unchanged. Any value to the left of W implies a likelihood ratio less than 1.0, and thus decreases the estimate of the probability that the screw is there. Both X and Y are to the left of W. Both suggest that the screw-is *not* there, but Y contributes more (as expected) toward decreasing the estimated probability that the screw is there than X does.

It is not true, however, that any value to the right of W increases the probability that the screw is there. Consider figure 4.1.8. It emphasizes the difference between the two

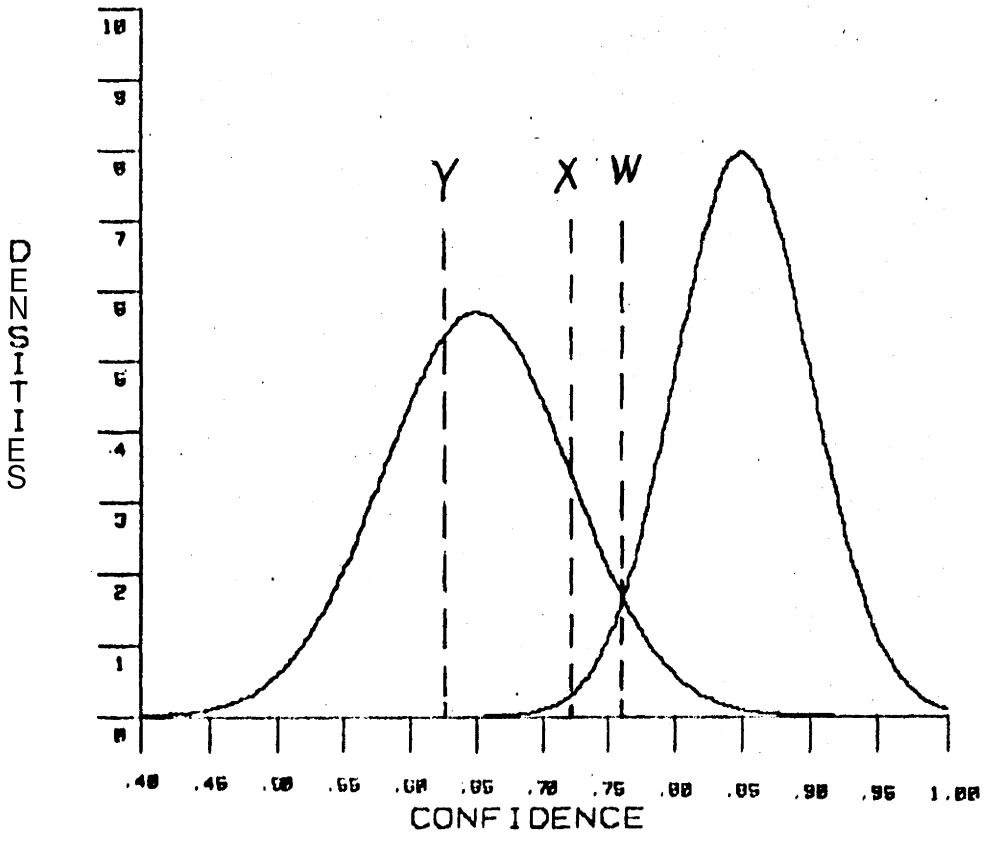

DENSITIES
10
9
8
7
6
5
4
3
2
1
0
Y
X
W
.40
.45
.50
.55
.60
.65
.70
.75
.80
.85
.90
.95
1.00
CONFIDENCE

FIGURE 4.1.7

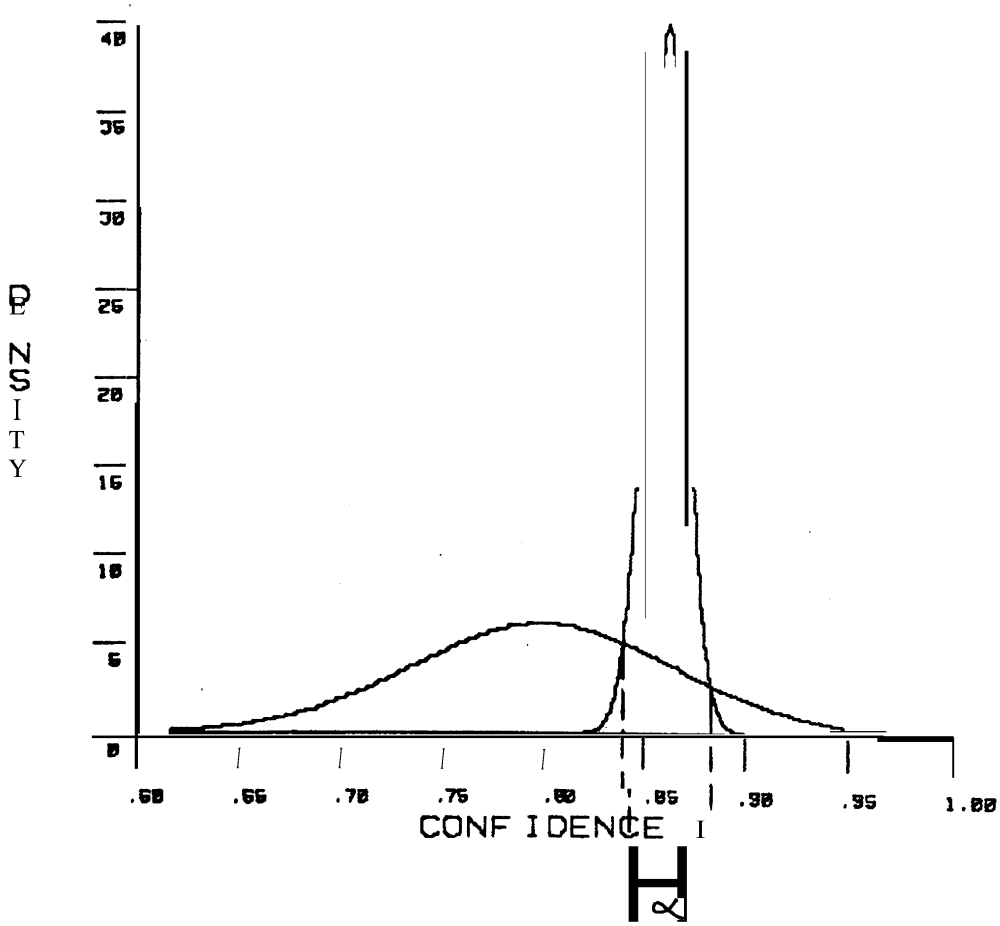
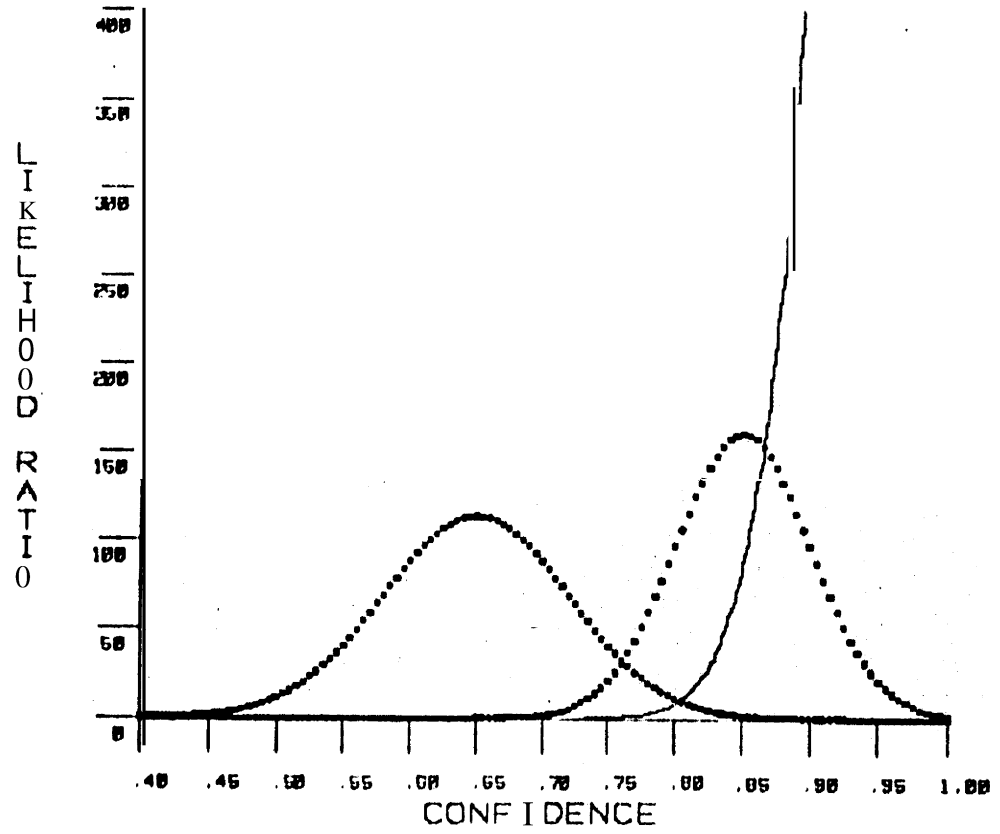

40
35
30
25
20
15
10
5
0
DENSITY
.60
.65
.70
.75
.80
.85
.90
.95
1.00
CONFIDENCE I
H
α

FIGURE 4.1.8


LIKELIHOOD RATIO
400
350
300
250
200
150
100
50
0
.40 .45 .50 .55 .60 .65 .70 .75 .80 .85 .90 .95 1.00
CONFIDENCE

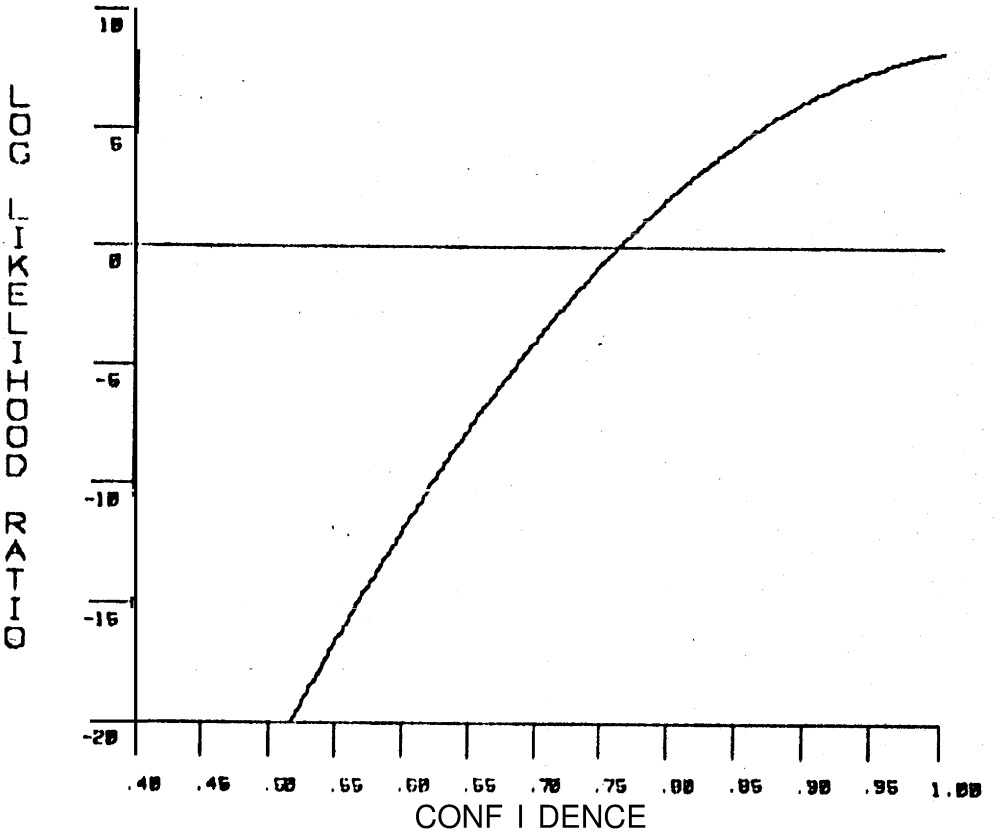

LOG LIKELIHOOD RATIO
10
5
0
-5
-10
-15
-20
.40 .45 .50 .55 .60 .65 .70 .75 .80 .85 .90 .95 1.00
CONFIDENCE

FIGURE 4.1.9

standard deviations so that it becomes clear that there is a region on the right of W in which the values produce likelihood ratios that are less than one. There is only a small interval (labeled a) that contains values that increase the probability that the screw is there. The likelihood ratios for every value in a are greater than one. All other values for the operator produce likelihood ratios less than one. Figure 4.1.9 shows the likelihood ratios and the log likelihood ratios associated with the distributions shown in figure 4.1.7.

This last form for **Bayes'** theorem is computationally convenient. For example, consider

(4.1.14)
$$t0 = \frac{P[\neg H]}{P[H]}$$

$$tN = \frac{P[vN|\neg H]}{P[vN|H]} * t(N-1) \qquad \text{(for } N > 0\text{)}$$

$$\text{and } P[H|v1,\ldots vN] = \frac{1}{1 + tN}.$$

This set of formulas gives a straightforward *way* of *incrementally* incorporating the results of sequentially applied, conditionally independent operators. In fact, it is a powerful way of combining the value information of operators into a probability that an object (or part of an object) is present.

Section 2
KNOWN ALTERNATIVES FOR A FEATURE

In the last section, an operator was applied over some portion of a picture under the assumption that there are only two possible results: (1) the screw is present and the operator locates the appropriate piece of the screw or (2) the screw is not present and the operator located some other "feature." It was also assumed that the operator was applied over the whole region before returning the "best" match. In effect, these assumptions guarantee that the value returned by the operator belongs to one of the two density functions (H or ¬H). This is pleasant if true, but there are-several reasons why these assumptions might be false:

(1) There may be other similar features in the same local area that sometimes

appear better to the operator than the "real" match. If a "similar" feature appears regularly enough so that the system can determine the corresponding density function, the feature will be called a *known alternative. In that* case the desired feature will no longer be special. It will simply be one of the alternatives. For any particular application of the operator, the system will have to decide which alternative is being matched. If a similar feature occurs infrequently and unpredictably, it will be referred to as a surprise.

(2) The measurements made by the operator may not immediately single out the "best" match. The value returned by a correlation operator orders the possible matches (the larger the correlation coefficient, the better). Other operators may return values along multiple scales, The "best" match is experimentally defined to be the one that produces values "closest" to the training values. For example, an edge operator may return (a) the distinctness of the edge and (b) the contrast-across the edge. If the "desired" line is a fuzzy line with a high contrast, it is not clear how to determine the "best" match. A distance function has to be defined.

(3) The "desired" feature may not be in the portion of the picture scanned by the operator. This problem may occur if the system has incorrectly narrowed down the set of possible positions for a match, or if the feature has been obscured for some reason. The operator will still return the location and value for the best match it can find, but it would be incorrect to assume that such a value belonged to one of the two densities and then draw some conclusions about the confidence of H.

(4) Some global factor may change (eg. one of the work station's lights may be out) so that the feature appears quite different, even though it is in the correct area.

(5) Each application of the operator may be so expensive that it is prohibitive to scan it over the complete area and choose the best match. Instead it has to be sequentially- applied until sdme "reasonably good" match is found. If there are a few similar features in the local area, a "reasonably good" match may not be the best match and hence the value produced by the operator may not belong to one of the two density functions.

This section develops the necessary mathematics to include *known alternatives* in the confidence computations.

Consider the problem of correctly deciding which one of three possible line segments an edge operator has located. There are several sources of information (orientation, fuzziness,

contrast, etc.), but for the time being only consider one dimension (eg. contrast). Assume that during the training session the system gathered enough statistics about the three lines to approximate the three density functions associated with their contrast values. If an edge is found with a certain contrast in an actual picture, which line is the operator on (assuming that there are only three possibilities) and what is the confidence associated with that decision? This question can be answered by computing three probabilities: the probability that the operator has located line 1, the probability that the operator has located line 2, and the probability that the operator has located line 3. Let

(4.2.1) L1 ≡ <operator 1 has located a point on line 1>
L2 ≡ <operator 1 has located a point on line 2>
and L3 ≡ <operator 1 has located a point on line 3>.

Then Bayes' theorem states that

$$P[L1|v] = \frac{1}{1 + \dfrac{P[v|\neg L1] * P[\neg L1]}{P[v|L1] * P[L1]}}. \tag{4.2.2}$$

Since

$$\neg L1 = L2 \oplus L3 \qquad \{ \oplus \text{ stands for exclusive OR } \}. \tag{4.2.3}$$

Then

$$P[v|\neg L1] = P[v|L2]*P[L2|\neg L1] + P[v|L3]*P[L3|\neg L1]. \tag{4.2.4}$$

Bayes' theorem reduces to

$$P[L1|v] = \frac{1}{1 + \dfrac{P[v|L2]*P[L2,\neg L1]}{P[v|L1]*P[L1]} + \dfrac{P[v|L3]*P[L3,\neg L1]}{P[v|L1]*P[L1]}} \tag{4.2.5}$$

or, since L2 and L3 are contained in ¬L1,

$$(4.2.6) \qquad P[L1|v] = \frac{1}{1 + \frac{P[v|L2]*P[L2]}{P[v|L1]*P[L1]} + \frac{P[v|L3]*P[L3]}{P[v|L1]*P[L1]}}.$$

When there are N known alternatives this becomes

$$(4.2.7) \qquad P[Lj|v] = \frac{1}{1 + \sum_{i \neq j} \frac{P[v|Li]*P[Li]}{P[v|Lj]*P[Lj]}}.$$

This formula is convenient because the desired probability is stated in terms of a priori probabilities and the simple ratios discussed in the last section.

The formula states how to compute the probability that the operator has located a particular feature, given several known alternatives. The alternative with the largest probability is the "best" match. Some best matches are better than others, however, in the sense that there is less chance of being *wrong.* For example, if there are four known alternatives, the system should. be more confident in its choice for the best match if the probabilities are P[L1|v] = .52, P[L2|v] = P[L3|v] = P[L4|v] = .16 than if the probabilities are P[L1|v] = .52, P[L2|v] = .46, P[L3|v] = P[L4|v] = .01, even though the best match has the same probability in both cases. One possible measure for this confidence is the ratio:

$$(4.2.8) \qquad \frac{P[\text{"best"} \mid v] - P[\text{"second best"} \mid v]}{P[\text{"best"} \mid v]}.$$

If the probability of the second best alternative is almost as large as the probability of the best alternative, the confidence will be low.

When the task is an *inspection-type* task (cg. checking to see if there is a screw on the screwdriver or not), there may be two or three known alternatives that are possible when the object is there and two or three known alternatives when the object is not there. In this case the system is less concerned with which alternative is the best match than it is with the overall probability that the object is there or not. A derivation similar to the one used above produces the formula needed in this situation. Let f1, f2, . . . fM be the known alternatives that might occur when the object is there and let g1, g2, . . . gN be the alternatives that are possible when the object is not there. Bayes' theorem states:

$$P[H|v] = \frac{1}{1 + \dfrac{P[v|\neg H] * P[\neg H]}{P[v|H] * P[H]}} \quad (4.2.9)$$

By assumption

$$P[H] = P[f1] + P[f2] + \ldots + P[fM] \quad (4.2.10)$$

and

$$P[\neg H] = P[g1] + P[g2] + \ldots + P[gN]. \quad (4.2.11)$$

Notice that this is equivalent to assuming that there are *no* surprises. Bayes' theorem can be expanded into

(4.2.12)

$$P[H|v] = \frac{1}{1 + \dfrac{P[v|g1]*P[g1] + P[v|g2]*P[g2] + \ldots + P[v|gM]*P[gN]}{P[v|f1]*P[f1] + P[v|f2]*P[f2] + \ldots + P[v|fM]*P[fM]}}$$

or

$$P[H|v] = \frac{1}{1 + \dfrac{\sum_{1 \le i \le N} P[v|gi]*P[gi]}{\sum_{1 \le i \le M} P[v|fi]*P[fi]}}. \quad (4.2.13)$$

In essence, this formula gathers all of the evidence for and against H and forms a ratio between them. To use this formula the system has to know a great deal about what can be expected in a **runtime** picture. In particular, the system must know what the possible alternatives are, what their values are, and how probable they are. Within the contest of programmable assembly this assumption is often reasonable because the environment is highly constrained and the system has the opportunity to watch several examples of the assembly.

This type of formula can be easily extended to incorporate the results of several operators, all of which may have known alternatives. Assume that there are K operators. Let **fj,1; fj,2;** . . . **;f j,** Nj be the N j known alternatives for the jth operator, when the

object is there. Let gj,1; gj,2; ...;gj,Mj be the Mj known alternatives for the jth operator, when the object is not there. Then

(4.2.14.)

$$P[H|v1,v2,...vK] = \frac{1}{1 + \frac{P[H]}{P[\neg H]}^{(K-1)} * \prod_{j=1}^{K} \frac{\sum_{1\le i\le Nj} P[vj|gj,i]*P[gj,i]}{\sum_{1\le i\le Mj} P[vj|fj,i]*P[fj,i]}}.$$

The exponent (K-I) appears because the expression for each of the K operators produces a factor of

(4.2.15)
$$\frac{P[H]}{P[\neg H]},$$

and the ratio of a *priori* probabilities in Bayes' theorem cancels one of them.

Section 3
SURPRISES

The main assumption of the last section was that all of the alternatives were known and characterized in advance. Sometimes, however, operators match unknown features and return unusual values. Such unknown and unexplained matches will be referred to as *surprises.* The values produced by surprises can not be accounted for by the usual density functions. There are two possible ways of dealing with these values: (1) filter out particularly bad values and (2) scale down the potential contribution (in the probability computations) of any operator that is known to find surprises. The first method involves a check on each value produced by an operator to make sure that it is a reasonable value for at least one of the known alternatives. For example, any value that is not within three standard deviations of the mean of a known alternative can be classified as an *unusual value.* There are several possible explanations for such a value (some global change, the feature is not present, or a surprise), but in any case the value should not be used to "improve" the confidence value. It may contribute to other considerations (such as some global error), but it should not be blindly cranked through the formula.

The second method lowers the possible contribution of the suspect operator because an operator that finds surprises should be trusted less than one that doesn't. The assumption used in the previous section that all of the alternatives are known is equivalent to the following equation relating the *a priori* probabilities:

(4.3.1) $$P[H] = P[f1] + P[f2] + \ldots P[fN].$$

If the operator occasionally locates surprises, a better model is

(4.3.2) $$P[H] = P[f1] + P[f2] + \ldots P[fN] + P[s]$$

where P[s] is the a *priori* probability of finding a surprise. To reflect this model in the probability computations requires some density function to be associated with the surprises. What should the form of this density be? If surprises can produce any value for the operator, one reasonable assumption is that the density is a rectangular distribution. And in light of the filtering mentioned in the last paragraph, it also seems reasonable to restrict the domain of this function to the interval between the smallest reasonable value for the operator and the largest reasonable value. Figure 4.3.1.a shows three density functions, one for the case when the screw is there and two known alternatives when the screw is not there. If the operator occasionally locates surprises, a rectangular density function is added, as shown in figure 4.3.1 .b.

The density function for surprises can be incorporated into the confidence computation in a straightforward way. Since a surprise may occur whether the object is there or not, the new possibility is included in both the numerator and the denominator. That is, if "s" represents the surprise, the formula is

(4.3.3) $$P[H|v] = \frac{1}{1 + \dfrac{P[v|s]*P[s] + \sum_{i=1}^{N} P[v|gi]*P[gi]}{P[v|s]*P[s] + \sum_{i=1}^{M} P[v|fi]*P[fi]}}.$$

The additional density function, therefore restricts the contribution of the suspect operator. The operator can not be as strongly for H or as strongly against H as it could when all of the alternatives were known. For example, if all of the P[v|gi]'s are essentially zero, the operator can no longer force the overall probability to one. The new addition also means that

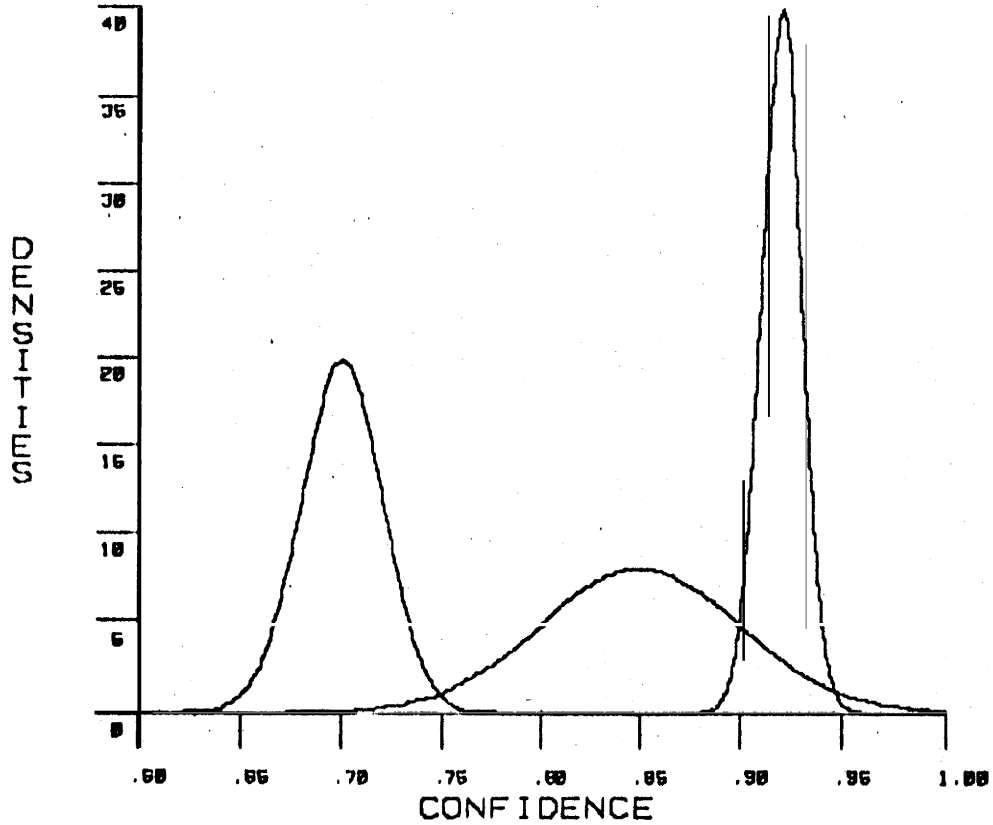


FIGURE 4.3. 1.a

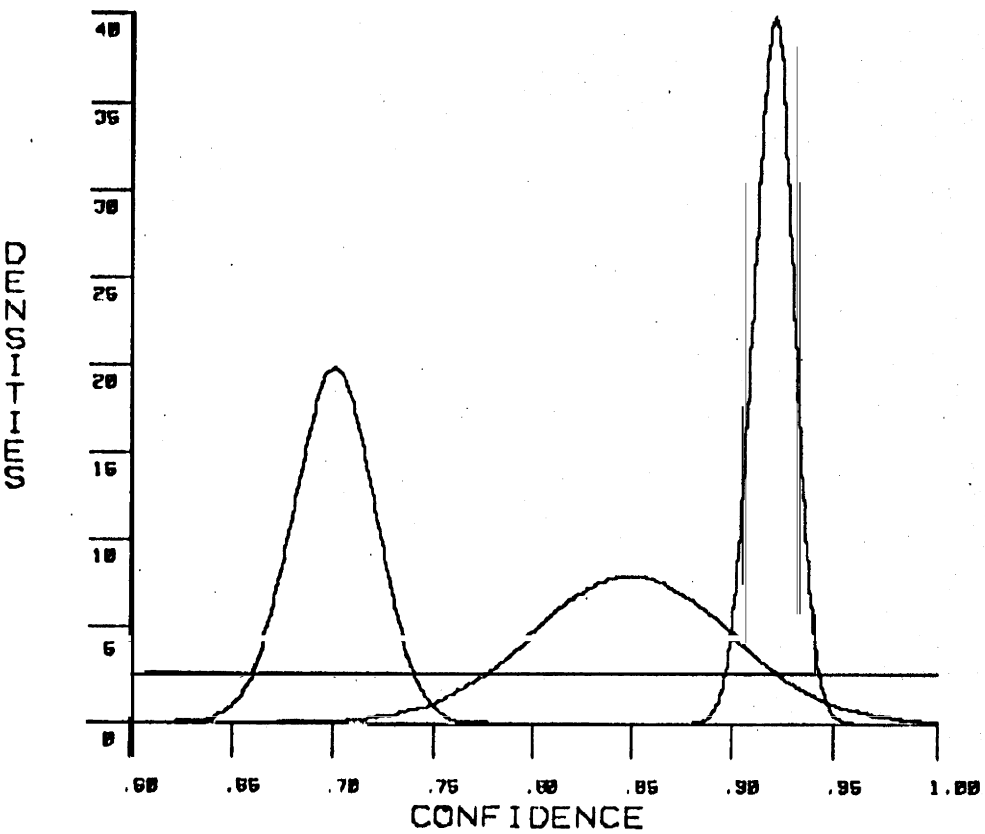


FIGURE 4.3.1.b

sometimes the "best" match will be the *surprise.* For example, if the operator happens to return the smallest reasonable value, the best match will probably be the surprise (depending upon the a *priori* probabilities).

If the operator happens to find surprises more often when the object is there (or not there), it is possible to set up two separate density functions and include one in the numerator and one in the denominator. Let $f_{j,0}$ be the surprise associated with the jth operator when the object is there and let $g_{j,0}$ be the surprise for the jth operator when the object is not there. Then the formula that combines the results of: several operators, each of which may have known alternatives and/or surprises, can be written as:

(4.3.4)

$$P[H|v1,v2,\ldots vK] = \frac{1}{1 + \frac{P[\neg H]}{P[H]} * \prod_{j=1}^{K} \left(\frac{P[H]}{P[\neg H]} * \frac{\sum_{0 \le i \le Nj} P[vj|gj,i]*P[gj,i]}{\sum_{0 \le i \le Mj} P[vj|fj,i]*P[fj,i]} \right)},$$

This extension of the formulas to include surprises means that there are three possible outcomes whenever an operator is applied: (1) the value is outside the "reasonable" range, (2) the value is reasonable, but it implies that the best match is a surprise, or (3) the value is reasonable and the best match is a known alternative. The interpretation, if any, of the unusual values and surprises has to be left up to a higher level system. A later chapter will pursue this question in more depth.

Section 4
MULTIPLE-VALUED OPERATORS

Some operators return more than one value; the description of what they have found contains values along several scales. For example, a texture operator may describe a local region in terms of many different characteristics. It has already been mentioned that edge operators-often return two or three values. When dealing with such operators one wants to combine all of the available information into one probability that the object is there, or to determine the best alternative. Again there are Bayesian probability formulas that provide one way of doing this. Consider an inspection task and one operator that returns N values,

v ,1 2,2... and vN. Then the standard Bayesian formula is

$$P[H|v1,v2,...vN] = \frac{1}{1 + \frac{P[v1,v2,...vN \mid \neg H]}{P[v1,v2,...vN \mid H]} * \frac{P[\neg H]}{P[H]}}. \quad (4.4.1)$$

If the values are conditionally independent of each other, the usual reduction yields

$$P[H|v1,v2,...vN] = \frac{1}{1 + \prod_{i=1}^{N} \frac{P[vi|\neg H]}{P[vi|H]} * \frac{P[\neg H]}{P[H]}}. \quad (4.4.2)$$

These formulas can be extended to include several operators, each of which may return several values. Assume that there are N operators and each operator returns M j values ($Mj \geq 1$). Let vj,1; v j,2; . . . vj,M j be the M j values returned by the jth operator. If the values for one operator are interdependent, but the values of separate operators are conditionally independen t, then

(4.4.3)

$$P[H \mid (vl,l;\ v1,2;\ \ldots\ v1,M1),\ \ldots\ (vN,1;\ vN,2;\ \ldots\ vN,MN)] =$$

$$\frac{1}{1 + \prod_{j=1}^{N} \frac{P[(vj,1;vj,2;\ldots;vj,Mj) \mid \neg H]}{P[(vj,1;vj,2;\ldots;vj,Mj) \mid H]} * \frac{P[\neg H]}{P[\neg H]}}.$$

If all of the values are conditionally independent of each other this formula collapses back into the previous formula (with a suitable renumbering of the v's).

This formula can be further extended to include operators that have several known alternatives and even surprises. Assume that the values for one operator are intcrdcpendcnt, butthat the values of separate operators are conditionally independent. Let there be K operators. Let the jth operator have M j known alternatives when the object is there, and N j known alternatives when the object is not there, and surprises. Assume that the jth operator returns R j values as a description of what it finds. Then the appropriate formula is:

(4.4.4)

$$P[H \mid (v1,1;v1,2;\ldots;v1,R1),\ \ldots\ (vN,1;vN,2;\ldots;vN,RN)] =$$

$$\frac{1}{1 + \frac{P[H]}{P[\neg H]} * \prod_{j=1}^{K} \frac{\sum_{i=0}^{Nj} P[(vi,1;vi,2;\ldots;vi,Ri) \mid gj,i]*P[gj,i]}{\sum_{i=0}^{Mj} P[(vi,1;vi,2;\ldots;vi,Ri) \mid fj,i]*P[fj,i]}}$$

To use operators that return several interdependent values the system has to gather enough information to approximate the multi-dimensional density functions. Once this has been done, the ratio of density values can be used in place of the ratio of probabilities, just as in the one-dimensional case.

Since the expression "(v j,1;. v j,2;... v j,R j)" can be validly substituted for "v j" in any of the derivations which follow, the remaining derivations will only be concerned with single-valued. operators. The formulas apply to multiple-valued operators, but for notational simplicity they will not be stated in their full generality.

Section 5
POSITION INFORMATION

The local value information produced by an operator is important, but the relative structure of the matches is crucial in verification vision. This section describes a method for incorporating the structural information into the relevant mathematical formulas.

Figure 4.5.1.a shows the positions of three typical features in a planning picture. Assume that the task involves determining the change from the planning picture to the actual picture and the change mainly consists of an X and Y shift. If the three operator-s are applied to an actual picture and the features are found at the positions shown in figure 4.5.1.b, a least squares fitting routine (or some other fitting routine) would be able to produce an estimate for the shift such that the errors between the actual locations for the matches and the predicted positions are quite small (as shown in figure 4.5.1.c). In this case one would

[V1.28]

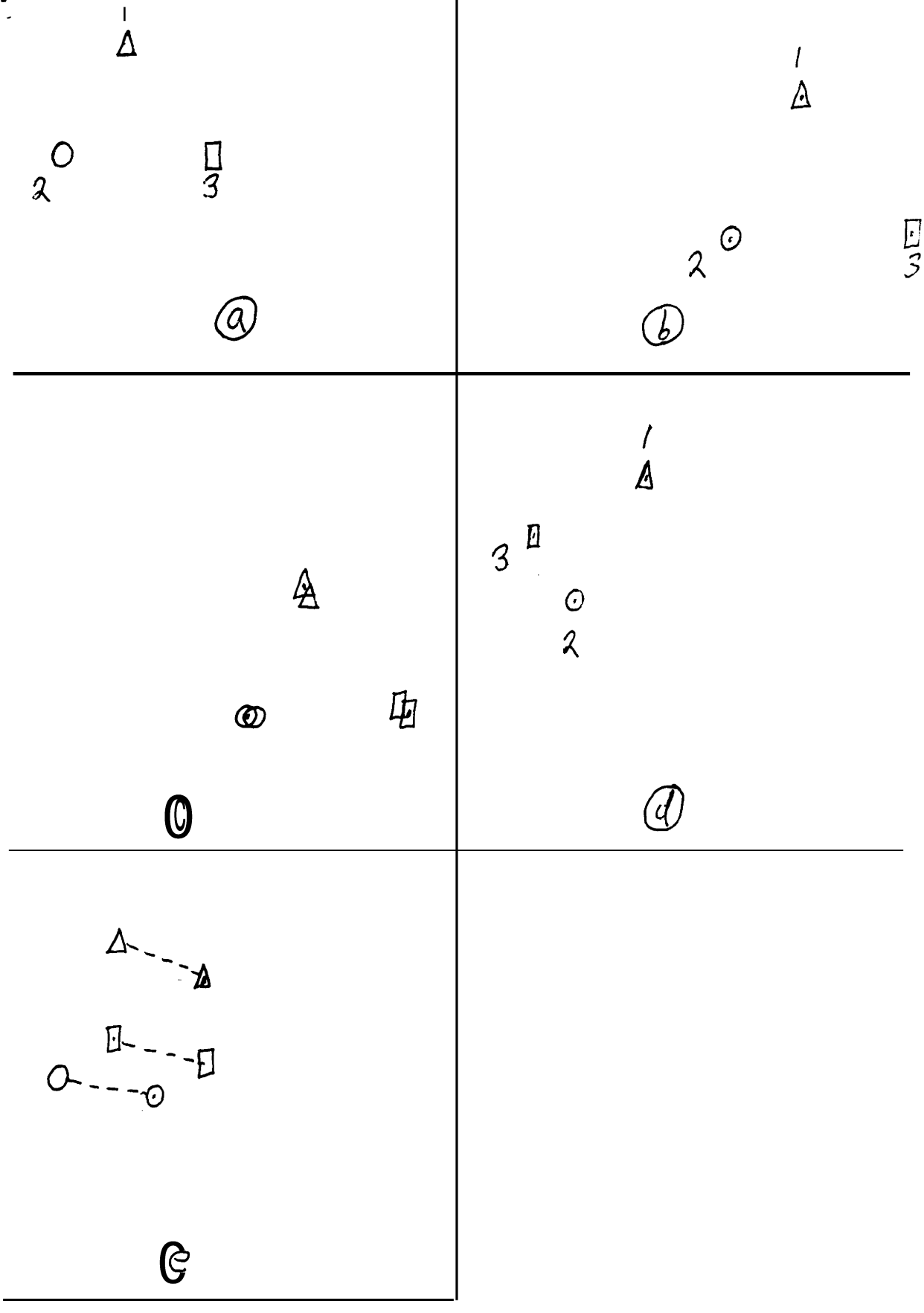


FIGURE 4.5.1

probably say that the operators are *structurally consistent*. However, if the three matches are found at the positions shown in figure 4.5.1.d, the best fit would still contain large errors (see figure 4.5.1.e). In this case one would probably be suspicious of at least one of the matches.

The implication is that the errors (remaining after a fitting routine has tried to determine the best transform that maps the planning positions of the features into their matching positions) are a function of the structural consistency of a set of matches. The less consistent the matches are, the larger the errors are. The sum of the squares of the errors is commonly used to measure this type of consistency. It is a convenient measure because there are well-known techniques for minimizing it. It is also appealing because the distribution for the sum of the squares of the errors is known to be a Chi-square distribution if the errors are normally distributed [ref LS book]. Since measurement errors are known to be normally distributed for a large number of situations, the use of least-squares techniques looks quite promising.

The theorem that specifies the distribution of the sum of the squares of the errors can be stated as follows:

> THEOREM: If there are N linear equations relating the actual matching positions with the planned positions and if there are R parameters to be adjusted in the transformation from the planned to actual positions, the sum of the squares of the errors (for normally distributed errors) forms a Chi-square distribution with (N-R) degrees of freedom.

This means that a Chi-square test can be applied to a particular sum of squares to determine whether it represents a consistent transformation between the planned and actual positions. If the test indicates that the set of matches is not consistent, it is possible to determine which match is' the least consistent. This least consistent match can be temporarily left out of the solution and another least squares fit can be computed; another test for consistency can be made, and so forth. This culling of "bad" matches can continue until a consistent set of matches has been found. Thus, another measure of the consistency of a set of matches is the percentage of matches deemed consistent by this culling procedure.

As expected, the concept of *structural consistency* is an important aspect of verification. The question is how to integrate it with the value information. Let

(4.52) Pi ≡ <operator i finds a match at position (x,y)>,

then Bayes' theorem becomes:

$$(4.5.2)\quad P[H|v1,\ldots vN,p1,\ldots pN] = \frac{1}{1 + \frac{P[v1,\ldots vN,p1\ldots pN|\neg H]}{P[v1,\ldots vN,p1\ldots pN|H]} * \frac{P[\neg H]}{P[H]}}.$$

If the vi's are assumed to be conditionally independent of the pi's (and each other), this reduces to:

$$(4.5.3)\quad P[H|v1,\ldots vN,p1,\ldots pN] = \frac{1}{1 + \prod_{i=1}^{N} \frac{P[vi|\neg H]}{P[vi|H]} * \frac{P[p1\ldots pN|\neg H]}{P[p1\ldots pN|H]} * \frac{P[\neg H]}{P[H]}}.$$

The assumption that the vi's are conditionally independent of the pi's means that the value of an operator is independent of the location of the match. That is, if the correct march is made, the value of the operator can be expected to be the same for all matching positions. This assumption is generally reasonable. However, if different positions consistently produce different lighting conditions (for example, cause a shadow to fall on a feature), the operator values may depend upon the position.

The assumption one does *not* want to make is that the Pi's are conditionally independent of each other. Such an assumption would completely ignore the structural consistency, which is precisely what the mathematics is intended to capture. But what is the value of

$$(4.5.4)\quad \frac{P[p1,p2,\ldots pN \mid \neg H]}{P[p1,p2,\ldots pN \mid H]} \; ?$$

It would be particularly hard to gather sufficient statistics in order to compute these probabilities directly. One heuristic that has proved to be experimentally useful is to replace this ratio by

$$(4.5.5)\quad \frac{\text{<percentage of consistent features, given } \neg H\text{>}}{\text{<percentage of consistent features, given } H\text{>}}.$$

This ratio does not really approximate the ratio of the probabilities, but it is useful because it provides a way of including a factor based upon the structural consistency of the matches.

In the simple case that each operator matches a *unique* feature when H is true, the system knows which feature to associate with each match. The least squares culling routine processes the list of pairs (planned feature position, matching position) and returns the number of consistent matches. Similarly, if each operator matches a *unique* feature when H is false, the system can construct the appropriate list of (planned position, actual position) pairs and determine the number of consistent matches. Since the total number of possible matches is the same for the two cases (H and ¬H) the ratio of percentages reduces to

$$\frac{\text{<number of consistent features, given } \neg H\text{>}}{\text{<number of consistent features, given } H\text{>}} \qquad (4.5.6)$$

Thus the contribution of "structural consistency" in the probability formulas has been transformed into a ratio of the numbers of consistent matches.

Recall that in the inspection-type tasks being described, the system does not know whether H is true or false, so it applies the same list of operators in both cases. The difference, of course, is that the operators will be matching different features in the two situations. The set of features for each situation (eg. ¬H) forms a geometric pattern (or structure). The structural consistency check involves assuming one such pattern, seeing how well it agrees with the resulting positions of the operators, and then trying the other pattern. The relative consistency of these two patterns determines the contribution toward the confidence of H.

In most cases the structure of the planning features when H is true is significantly different from the structure of the planning features when H is false. This guarantees that the ratio will seldom be close to 1.0. Intuitively this result is correct because it would be surprising for the operators to find their best matches in both cases (H and ¬H) in such a way that they-formed the same geometric pattern..

An important assumption of this discussion is that the operators match unique features, one for H and one for ¬H. In order to apply the least squares culling routine the system needs to know which feature on the object to associate with each match. If the system does not know which features are being matched, it has no way of knowing what the structure of the matches should be or how consistent the set of matches is.

If there are several known alternatives, the system can use the alternative with the highest probability of being the correct match. Recall that the basic formula used to determine-the best alternative is

(4.5.7)
$$P[Lj|v] = \frac{1}{1 + \sum_{i \neq j} \frac{P[v|Li]*P[Li]}{P[v|Lj]*P[Lj]}}.$$

If there happen to be two or more alternatives that have approximately equal probabilities of being the "best" match, the least squares culling procedure can be extended as follows: whenever the first choice is about to be discarded (because it is the least consistent match), another approximately equal choice can be tried in its place. This increases the complexity of the least squares culling routine, but it provides an automatic way of giving an operator the necessary second chance whenever there is more than one possible explanation for its results.

'The incorporation of the position information does not alter the ease with which the probabilities can be computed. Sequentially acquired information can still be included very nicely. Since the least squares culling procedure can not be applied until some minimum number of features has been located, the position information can not contribute anything until then. The minimum number depends upon the number of parameters being adjusted, the number of equations contributed by each feature, and any independence conditions. For example, if the least-squares method is performing a planar fit, there are three parameters, dX, dY, and $d\alpha$. Since each correlation feature and each point-on-a-line feature contributes two equations, any two of these features would be sufficient. Three or four would be better because the least squares technique works better when the parameters are over-constrained. Since this is true, the system may chose more than the minimum number of features before trying to incorporate the position information.

If there are several known alternatives for each feature, each operator does not necessarily contribute one "good" match toward the minimum needed to incorporate the position information. A better estimate is the probability associated with the best match. Thus, if the probability of matching one of the alternatives is .8, it must be the best match, and the operator contributes .8 of a "good" feature toward the desired minimum.

Figure 4.5.2 outlines the general method suggested by this section. One operator after another is applied until the accumulated value information indicates that sufficient features have been located; then the least-squares method is applied. Additional features are added until the confidence reaches the desired limit. This algorithm could form the basis for a "discrete inspection" system. It could be used to check to see if a gasket is already on or not, if a hole has been drilled or not, or if the expected subassembly has been added.

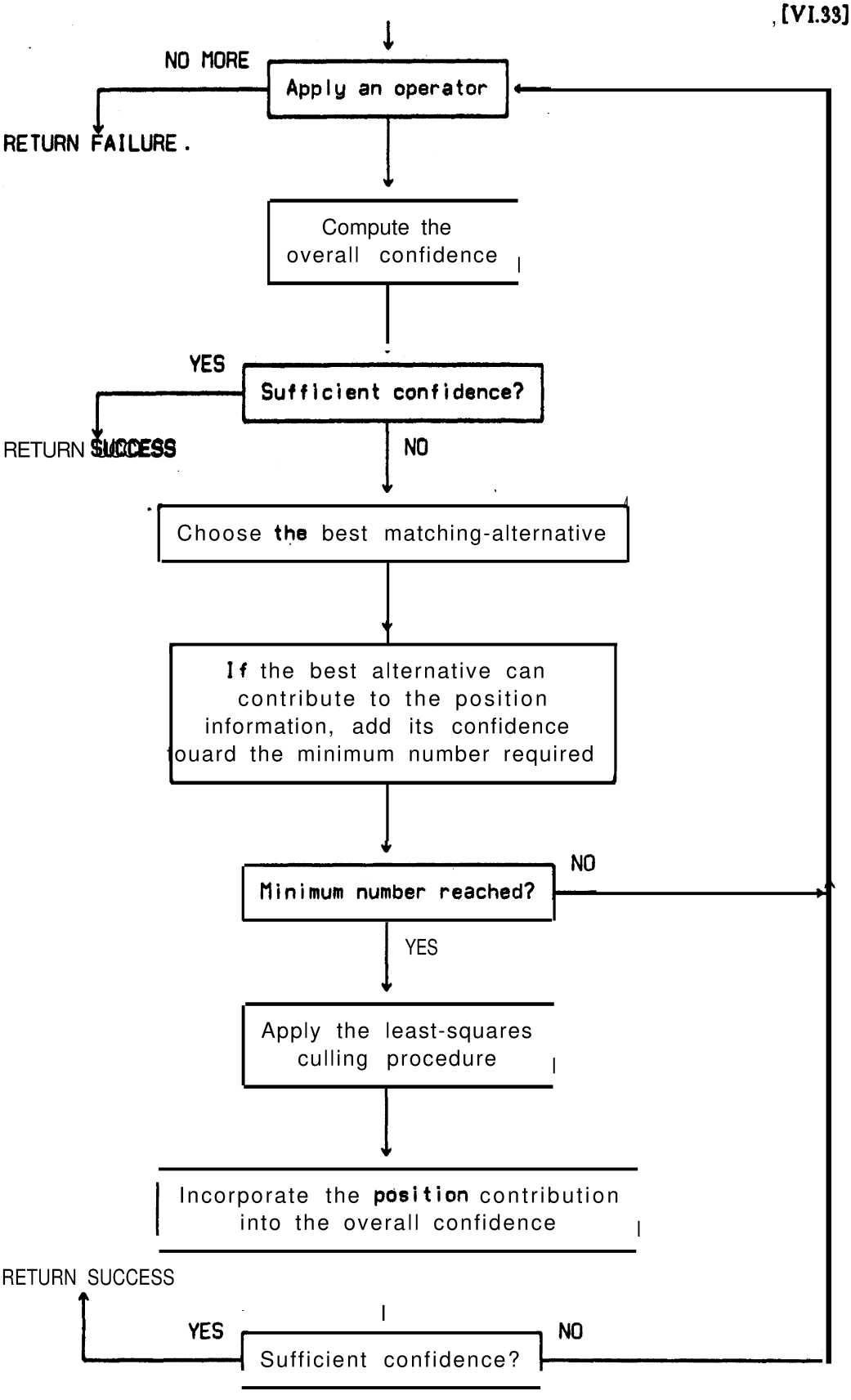

NO MORE
Apply an operator
RETURN FAILURE.
Compute the overall confidence
YES
Sufficient confidence?
RETURN SUCCESS
NO
Choose the best matching-alternative
If the best alternative can contribute to the position information, add its confidence
ouard the minimum number required
NO
Minimum number reached?
YES
Apply the least-squares culling procedure
Incorporate the position contribution into the overall confidence
RETURN SUCCESS
YES
NO
Sufficient confidence?

FIGURE 4.5.2

CHAPTER 3

EXECUTION-TIME MATHEMATICS FOR LOCATION

If the verification vision system is trying to locate, not inspect, an object, there are two important parameters: (1) an estimate for the object's location and (2) the precision associated with that estimate. In the context of VV the *location* of *an object* refers to the position and orientation of the object's coordinate system in terms of some other coordinate system (eg. the work station's coordinate system). Usually there is some point on the object of particular interest; eg. the center of a hole or the tip of a screw. Such a point will be referred to as a *point of interest.*

The last chapter briefly mentioned that a least-squares method conveniently combines a set of planned positions with a set of corresponding measured positions to produce an estimate for the transformation between them. Given this transformation and the planned position of the object, it is easy to compute the current estimate for the object's location. The least-squares technique can also produce the standard deviations associated with the estimates for the individual parameters in the transform. These standard deviations can be combined to produce an estimate for the precision.

The application of the least-squares technique depends upon knowing the correspondence between the matching points and the planning features, If the correspondence is correct, the estimate for the object's location and the associated precision will be correct. However, it is possible for an incorrect correspondence to lead to a (seemingly) *structurally consistent* subset of the features, which leads in turn, to an incorrect estimate for the object's location. This problem only arises when there are several known alternatives for the features or when the operators find surprises. To avoid incorrectly reporting a location it is necessary to incorporate the operators' value information with the least-squares information to produce an overall probability that the object is within the stated precision of the estimate. This chapter begins with a detailed explanation of how a least-squares method can be applied to the VV problem to produce a location and a precision. The second section describes a situation in which the results of the least-squares method are incorrect and then presents a simple method for producing a rough estimate for the confidence associated with a statement of precision.

Section 1
DETERMINING PRECISION

This section presents a general method for performing *nonlinear* generalized least-squares adjustments. A major portion of this discussion is a restatement of an internal paper at the Stanford Artificial Intelligence Project written by Donald B. Gennery entitled "Least-Squares Stereo~Camera Calibration." The method uses partial derivatives to approximate the problem under the general linear hypothesis model of statistics, and then iterates to achieve the exact solution. For more detailed information see [Graybill61].

The notational conventions are the following. Capital letters denote matrices. Vectors are represented by column matrices. A particular element of a matrix is represented by the corresponding lower-case letter followed by the appropriate indices. The transpose of a matrix A is denoted by A', and the inverse of A is denoted by A~. Multiplication (either scalar or matrix) is denoted by an asterisk.

Let the vector G denote a set of m unknown parameters for which values are desired. Let the vector U be a set of n scalar quantities ($n \geq m$) that are functions of G and can be measured *with some* error. Let F represent the vector of n functions that relate elements in U with G. Given an estimate for G, F(C) produces an estimate for U. Finally let the vector V represent the n residuals (ie. the unexplained errors) that remain between U and an estimate produced by F(G). Thus

(5.1.1) $$U = F(G) + V.$$

The goal is to eliminate (or minimize) V by modifications to C.

In verification vision G is the set of parameters in the transform that maps the planned positions of the features into their matching positions (ie. the planned positions into the measured positions). Typical elements in G are the displacement in X (dx), the displacement in Z (dz), and the unknown rotation about the Z-axis (da). Different features contribute different components to U and F. For example, when the transform is planar (so that the unknown parameters are dx, dy, and da), a correlation feature contributes two measured values to U: the X and Y components of the match (let. them be referred to as Xm and Ym). The corresponding functions in F are:

(5.1.2)
$$Xe = (Xp-Xc)*COS(d\alpha) - (Yp-Yc)*SIN(d\alpha) + dx + Xc$$
$$Ye = (Xp-Xc)*SIN(d\alpha) + (Yp-Yc)*COS(d\alpha) + dy + Yc$$

where (Xc,Yc) is the center of rotation for dα, (Xp,Yp) is the planned position for the correlation patch, and (Xe,Ye) is the transformed position of (Xp,Yp). The transformed position of (Xp,Yp) is the estimate for (Xp,Yp)'s position in the current picture. The two residuals that would be associated with a correlation feature are

(5.1.3) $$Xm - Xe$$
$$\text{and } Ym - Ye.$$

These residuals are the components of V. The goal, of course, is to use the measured values to improve the estimates for the parameters,

The quadratic form

(5.1.4) $$q = V'*W*V$$

is the criterion of optimization that is to be minimized. W denotes an n by n weight matrix. If W is the inverse of the covariance matrix of the errors in the observations, the result will be the maximum likelihood {in the F space) solution if the errors have a normal distribution. If W is a diagonal matrix, which indicates no correlation between errors in the different observations, the quadratic form reduces to a weighted sum of the squares of the elements of V. Thus the problem as stated here can be said to be a generalized least-squares adjustment.

The difficulty in obtaining a solution to the above problem lies in the fact that F in (5.1.1) is a nonlinear function, and thus in general there is no closed-form solution. One way of solving the problem is to use some type of general numerical minimization technique, which tries new values of G, recomputes q, and tries to drive q to a minimum. However, such methods tend to converge rather slowly. Also, numerical problems may occur if q has a very broad minimum, for round-off errors may give rise to spurious local minima. Instead of such an approach, the method described here approximates (5.1.1) by a linearization based on the partial derivatives of F, solves the resulting linear problem, and iterates this process to obtain the solution to the nonlinear problem.

Let the n by m matrix P be composed of the partial derivatives of the functions in F, such that

(5.1.5) $$pij = \frac{\partial fi}{\partial gj}.$$

Let GO denote an approximation to G. Then equation (5.1.1) can be approximated as follows:

(5.1.6) $$U = F(G0) + P(G0)*(G - G0) + V$$

where the functional dependence of P on G has been explicitly indicated. Define

(5.1.7) $$E = U - F(G0)$$
$$D = G-G0.$$

Then (5.1.6) can be rewritten as

(5.1.8) $$E = P*D + V .$$

Thus the nonlinear equation (5.1.1) has been replaced by the linear equation (5.1.8), in which E represents the discrepancy between the observations and their computed values (using the current approximations of the parameters), and D represents the corrections needed to the parameters.

It is necessary to solve for D in (5.1.8) in order to minimize q in (51.4). This is a standard problem in linear statistical models (eg. see [Graybill71]). The solution for D is

(5.1.9) $$D = (P'*W*P)\sim*(P'*W*E)$$

and the covariance matrix of errors in the solution for D is

(5.1.10) $$S = (P'*W*P)\sim$$

assuming that W is the inverse of the covariancc matrix of the observation errors.

Several other quantities of interest can be derived from the solution. The expected value of q is n-m. If the scale factor of the covariance matrix of observation errors is unknown, W can be adjusted by the ratio (n-m)/q and S by the ratio q/(n-m). Otherwise, q can be used as a test on the adjustment; for, if the observation errors have the Gaussian distribution, q has the chi-square distribution with n-m degrees of freedom. S represents the covariance matrix of errors in the adjusted parameters. The square roots of the diagonal elements of S are the standard deviations of the adjusted parameters. The correlation matrix of the parameters can be obtained from S by dividing the i,j element by the product of the standard deviations of the ith and jth parameters, for all i and j.

Other results are the covariance matrix of the adjusted observations P*S*P' and the covariance matrix of the residuals W~ - (P*S*P'). It is often useful to compare the magnitude of the residuals to their standard deviations, ie. the square roots of the diagonal elements of their covariance matrix. If a residual is greater than two (or three) standard deviations it indicates that the associated measured value is "inconsistent" with the other

values used to compute the estimate for the transform. This test is the basis for the least-squares culling procedure mentioned in the previous chapter.

The covariance matrix about a point not in the solution is $W\sim + (P*S*P')$ where P is the set of partial derivatives at the point and W is the inverse of the covariance matrix that weights the measured values. In VV the standard deviation that can be computed from this covariance matrix can be used to determine the uncertainty associated with any other point on the ob jcct (cg. a point *of interest*). It can also be used to determine the tolerance region about the next feature to be tried.

The solution of the nonlinear problem can now be described as follows. An initial approximation is used to compute the discrepancies Ei and the partial derivatives Pij. Then D is computed from (5.1.9) and is added to the current approximation for C to obtain a better approximation. This process repeats until there is no further appreciable change in C. Then the final values from the last iteration can be used to obtain S, Vi, q, and the other derived quantities described above. Of course, in order to converge to the absolute minimum of q rather than convergence to some local minimum or divergence, it is necessary that the initial approximation be sufficiently close to the true solution. In most practical problems the initial approximation is not critical; in fact, often there is only one minimum.

Since on the last iteration the partial derivatives have been computed for the converged value of C, the solution gives the true generalized least-squares adjustment regardless of the nonlinearity, However, some of the other properties of the adjustment arc only approximate in the nonlinear case. Among these are the use of S as the covariance matrix of the errors in the final value of C, and the properties that the solution for C is minimum-variance and unbiased. However, if the amount of nonlinearity *over* the range of the measurement errors is small, these results will be fairly accurate.

A few comments should be made about the numerical aspects of performing the computations. The H matrix is always non-negative definite; that is, if it is not singular it is positive definite. The best strategy to use when inverting a positive-definite matrix by an elimination technique is to pivot on the main diagonal (see [Forsythe 71]). Therefore, a simple matrix inverttr without any pivoting can-be used to obtain $H\sim$. H is also symmetrical; therefore, some computation time can be saved if the inverter makes use of this fact. However, if n is considerably larger than m, much more time is spent in computing H than in inverting it, so this special care is hardly worth the trouble. In problems where the solution is nearly indeterminate, H will be nearly singular, and much accuracy can be lost because of numerical roundoff error. In such cases it may be necessary to use double precision in the computations for H, C, D, and S according to (5.1.9), and for the inversion of H. (If a good inverter is used, there is usually not much point in having it in double precision unless a double-precision H is available to invert, as explained in [Forsythe 71].) However, high precision is not needed in computing the discrepancies Ei and the partial derivatives Pi, as

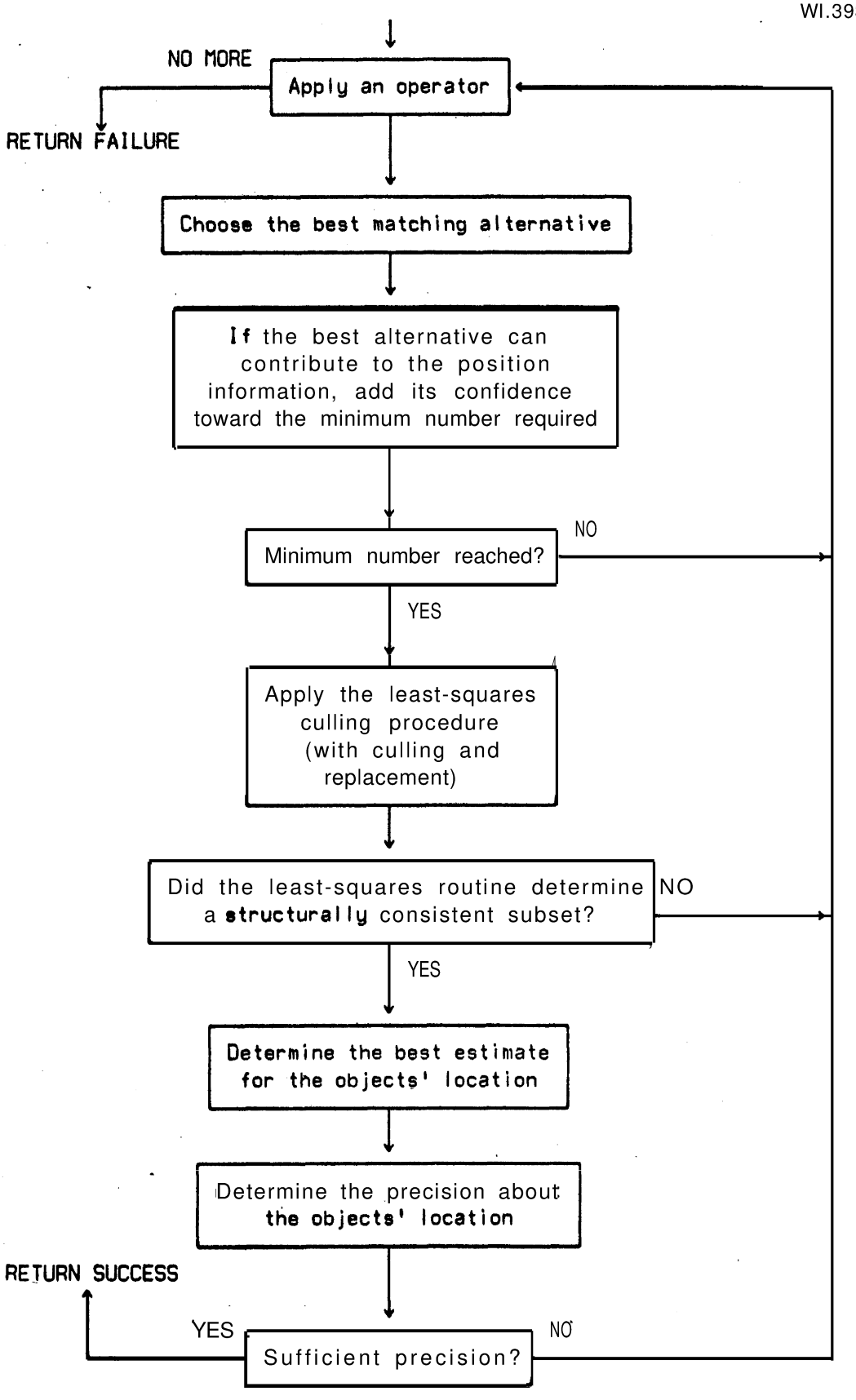

NO MORE
Apply an operator
RETURN FAILURE
Choose the best matching alternative
If the best alternative can contribute to the position information, add its confidence toward the minimum number required
Minimum number reached?
NO
YES
Apply the least-squares culling procedure (with culling and replacement)
Did the least-squares routine determine a structurally consistent subset?
NO
YES
Determine the best estimate for the objects' location
Determine the precision about the objects' location
RETURN SUCCESS
YES
Sufficient precision?
NO

FIGURE 5.1.1

long as consistent values are used throughout the computations for H and C.

Figure 5.1.1 is a flowchart that outlines the basic steps involved in using a least-squares method to compute an estimate for an object's position and a precision about that estimate. The algorithm is a sequential algorithm that applies the least-squares routine as soon as a sufficient number of features has been found. The best values for the parameters are used to map the object's planned location into an estimate for its current location. The standard deviations associated with the best parameter values are combined to produce a region of uncertainty about the estimate. As stated, the algorithm is concerned with the object's location. Given the object's estimate and precision, it is easy to produce estimates and uncertainty regions for any other points *of interest* on the object.

Section 2
CONFIDENCE IN THE PRECISION

The algorithm shown in figure 5.1.1 can be used by itself to locate objects. However, to do so requires an assumption: if the least-squares culling routine determines a structurally consistent subset of the features, and if the desired precision has been reached, then a correct correspondence has been established between the positions produced by the operators and the known alternatives for the features. This assumption is generally reasonable when the number of known alternatives is small and the operators are reliable (ie. they do not locate surprises very frequently). However, it is possible to locate a set of features that appears (to the least-squares culling routine) to be structurally consistent, when in fact, some of the results have been incorrectly associated with alternatives. For example, consider figure 5.2.1. Figure 5.2.1.a shows a point of *interest* and a set of known alternatives for four operators. Operators three and four can each find two known alternative features. Figure 5.2.1.b shows the *actual* positions of all of these points in a particular runtime picture. These positions are *not* the positions where the operators found them, but the positions where the operators should have found them, if the operators were reliable. Figure 5.2.1.c superimposes the four positions where the operators think they have located known alternatives on top of the actual positions. So far the operators are correct. However, if the system decides that operator three has matched alternative 3.a and that operator four has matched alternative 4.a (both of which are wrong), the least-squares routine will probably decide that the features are structurally consistent and proceed to place the estimate for the point of interest at the position shown in figure 5.2.1.d. This conclusion is wrong. The cause of this error was the system's incorrect assignment of alternatives to the operators' results. The resulting assignment happens to appear to be structurally consistent and the system, having fooled itself, proceeds to draw an incorrect conclusion. This example is a simple example, but it points out a potential danger

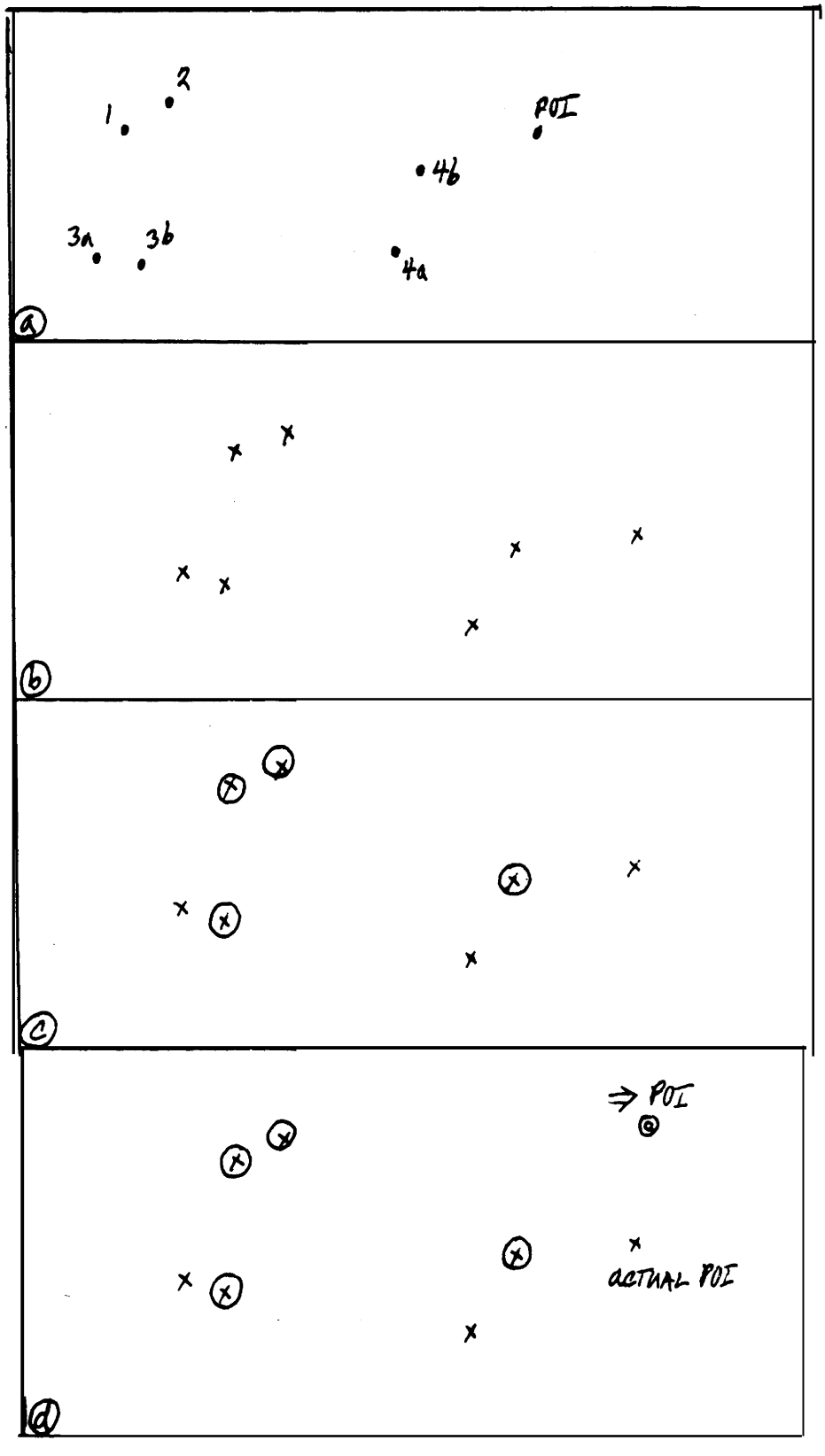


FIGURE 5.2.1

in unconditionally believing the results of the least-squares culling routine.

One way to avoid this type of incorrect deduction is to use more and more features, which makes it less and less likely that the results will be incorrectly judged to be structurally consistent. This solution is fine if the features are inexpensive to apply. However, if the system cries to minimize the number of features applied and minimize the amount of work required to locate each match, some measure of the confidence in the precision produced by the least-squares routine is necessary. This confidence helps the system minimize the number of features by allowing the system to stop applying features as soon as the desired precision and confidence in the precision have been met. It helps to minimize the amount of work required because it provides a way of deciding when a precision can be *safely* used to restrict the region that should be scanned in order to locate a new feature.

Unfortunately, it is difficult to compute the probability that the object is actually within the stated precision of the least-squares' estimate of its location. The computation requires several new assumptions. There are, however, some *ad hoc,* but experimentally useful methods for developing an estimate for the confidence.

One crude measure of the confidence associated with an assignment is the average of the probabilities associated with the individual matches:

$$\frac{\sum_{1 \le i \le N} P[f_{i,m_i} \mid v_i]}{N} \qquad (5.2.3)$$

where f_{i,m_i} is the known alternative chosen by the least-squares culling routine as the match for operator i. The higher the average, the more confidence there is in the assignment. The average of the individual matches has the nice property that one uncertain match can be "averaged out" by' several distinctly matched features. This property is nice because it means that one (or two) dubious matches can be part of an assignment without drastically lowering its confidence. However, in conjunction with this, it is not possible co combine several *reasonably distinct* matches into an assignment with a very h'igh overall confidence.

This type of measure is really only a general indicator of the overall confidence that should be associated with an assignment. It does not approximate the probability that the object has been correctly located.

CHAPTER4

PLANNING-TIMEMATHEMATICS

The goal of this chapter is Co investigate ways of producing information that is useful to a *strategist*. In this context a strategist is a program (or possibly a person) that evaluates the various alternatives and develops a plan Co achieve a particular goal. At one level a strategist might be trying Co decide whether Co use visual feedback or force feedback to check for a screw on the end of the screwdriver. Ac that-decision point it needs information about the expected costs and reliabilites of the alternative methods (see [Taylor 76] and [Sproull 76] for descriptions of strategists and the information used Co make decisions). This chapter develops techniques. for producing this type of cost arid reliabilty estimates for verification vision.

Execution-time mathematics provides methods for combining the results of sequentially applied operators to produce estimates for inspection confidences, precision, and precision confidences. These methods make it possible for the system to stop gathering information as soon as the desired confidence and precision have been reached. The underlying technique is an *ordered* list of operators to be tried. The ordering criteria are important because some operators are more reliable than others, some contribute more than others, and some operators cost more to apply than others. This chapter investigates techniques for ordering the operators according to their *expected* contributions and costs. It also presents techniques for estimating the *expected* number of features (and costs) required co achieve certain coniidcnce and precision limits.

The first few sections describe the mathematical tools used to rank operators by value and cost estimates. The last few sections develop techniques co predict the *expected* number of features necessary to reach various limits like the minimum number of features required to apply the least-squares culling routine.

Section I
RANKING FEATURES BY VALUE

Consider the task of inspecting a scene to decide whether a screw is present or not. Section 4.1 developed a formula that reduces the value information from several operators into an overall confidence that the object is present. It also pointed out that the *contribution* of an operator is the value of the ratio:

$$\frac{P[vi|\neg H]}{P[vi|H]} \tag{6.1.1}$$

where vi is the value (or set of values) returned by the operator and H denotes the proposition that the object is there (SEC formula 4.1.12). For ranking purposes the logarithm of the inverse ratio is more convenient:

$$\log\left(\frac{P[vi|H]}{P[vi|\neg H]}\right). \tag{6.1.2}$$

The greater the ratio, the better the contribution. The logarithm of the likelihood ratio is used because there is a theorem (to be discussed in section 6.5) that shows how to compute an estimate for the number of operators required to reach a 'certain confidence from the log-ratios of the operator&

At planning time, vi does not yet have a specific value, so the system is interested in the average (or expected) value of this log-ratio. To compute this expected log-ratio one needs the density function for vi, which is a weighted sum of the density functions for H and ¬H (as shown in figure 4.1.6). The weights are simply the a *priori* probabilities. Therefore,

(6.1.3) **density(vi) = P[H] * H_density(vi) + P[¬H] * ¬H_density(vi).**

This is a valid density function since

(6.1.4)

$$\int_{-\infty}^{+00} \text{density}(X)dX = \int_{-\infty}^{+00} P[H]*H\_\text{density}(X)dX + \int_{-\infty}^{+\infty} P[\neg H]*\neg H\_\text{density}(X)dX$$

$$= P[H]*\int_{-\infty}^{+\infty} H\_\text{density}(X)dX + P[\neg H]*\int_{-\infty}^{+\infty} \neg H\_\text{density}(X)dX$$

$$= P[H] + P[\neg H] = 1 .$$

The expected value can then be computed as follows:

(6.1.5) $$\text{expected-log-ratio} = \int_{-\infty}^{+\infty} \text{log-ratio}(X)'* \text{density}(X)\ dX.$$

MAXSYMA (see [MAXSYMA **ref**]) was used to expand this integral symbolically, assuming that the density functions are normal. The derivation is given in the appendix. The result is a readily evaluated expression of the two means (M1 & M2), the two standard deviations (SD1 & SD2), and the *a priori* probability of H (ie. P):

(6.1.6) $$\text{expected-log-ratio} = \log(SD2) - \log(SD1) + 1/2 - P$$

$$+ P*\frac{SD1^2 + (M2 - M1)^2}{2*SD2^2} - (1-P)*\frac{SD2^2 + (M2-M1)^2}{2*SD1^2}.$$

Later sections will also need estimates for the expected log-ratio, given either H or ¬H. The expected log-ratiq, given **H,** can be computed as follows:

(6.1.7) $$\text{ELR\_given\_H} = \int_{-\infty}^{+\infty} \text{log-ratio}(X) * \text{H-density}(X)\ dX.$$

The integral can be expanded to produce

$$(6.1.8)\quad ELR\_given\_H = \log(S02) - \log(SD1) + \frac{SD1^2 + (M2 - M1)^2}{2 * so2^2} - \frac{1}{2}.$$

Similarly, the expected log-ratio, given ¬H, can be expressed as

$$(6.1.9)\quad ELR\_given\_\neg H = \log(S02) - \log(SO1) + \frac{1}{2} - \frac{SO2^2 + (M2 - M1)^2}{2 * SD1^2}.$$

Since the expected log-ratio for an operator represents the operator's average contribution, operators that have large expected log-ratios should be applied first in order to minimize the number of operators used to reach some confidence limit. Thus, a simple operator-ranking scheme consists of computing the expected log-ratio for each of the operators and then ordering them according to their expected value (largest first).

Section 2
KNOWN ALTERNATIVES AND SURPRISES

The method used in the last section can be used to compute the expected contributions for operators that have several *known alternatives* and/or are subject to *surprises*. However, it is quite difficult to expand symbolically the integrals that express the expected value. A numerical technique is used instead.

Formula (4.3.4) expresses the probability that the object is present given the values of several operators, each of which may have several known alternatives and surprises. That formula is

(6.2.1)

$$P[H|v1,v2,\ldots vK] = \frac{1}{1 + \frac{P[\neg H]}{P[H]} * \prod_{j=1}^{K} \left(\frac{P[H]}{P[\neg H]} * \frac{\sum_{0 \le i \le Nj} P[vj|gj,i]*P[gj,i]}{\sum_{0 \le i \le Mj} P[vj|fj,i]*P[fj,i]} \right)},$$

where $fj,1; fj,2; \ldots fj,Nj$ are the Nj known alternatives for jth operator when H is true, $gj,1; gj,2; \ldots gj,Mj$ are the Mj known alternatives for jth operator when H is false, $f_{j,0}$ is the surprise for jth operator when H is true, and $g_{j,0}$ is the surprise for jth operator when H is false. The *contribution* of the jth operator toward the overall probability is:

(6.2.2)
$$\left(\frac{P[H]}{P[\neg H]} * \frac{\sum_{0 \le i \le Nj} P[vj|gj,i]*P[gj,i]}{\sum_{0 \le i \le Mj} P[vj|fj,i]*P[fj,i]} \right).$$

For ranking purposes the logarithm of the inverse of this ratio is used:

(6.2.3)
$$\text{log-ratio}(vj) = \log\left(\frac{P[\neg H]}{P[H]} * \frac{\sum_{0 \le i \le Mj} P[vj|fj,i]*P[fj,i]}{\sum_{0 \le i \le N} P[vj|gj,i]*P[gj,i]} \right).$$

The expected value can again be computed by

(6.2.4)
$$\text{expected-log-ratio} = \int_{-\infty}^{+\infty} \text{log-ratio}(X) * \text{density}(X)\, dX,$$

where the density depends upon all of the known alternatives and surprises. Since

(6.2.5)
$$P[H] = P[fj,0] + P[fj,1] + \ldots P[fj,Nj]$$
$$\text{and } P[\neg H] = P[gj,0] + P[gj,1] + \ldots P[gj,Mj],$$

the density for operator **j** is

(6.2.6)

$$\text{density}(X) = \sum_{i=0}^{Mj}(P[fj,i]*\text{density}(fj,i)) + \sum_{i=0}^{Nj}(P[gj,i]*\text{density}(gj,i)).$$

Thus, if ELR denotes the expected log-ratio for the jth operator, then

(6.2.7)

$$ELR = \int_{-\infty}^{+\infty} \log\left(\frac{P[\neg H]}{P[H]} * \frac{\sum_{0\le i\le Mj} P[vJ|fj,i]*P[fj,i]}{\sum_{0\le i\le Nj} P[vJ|gj,i]*P[gj,i]}\right) * \text{density}(X)dX$$

or

(G.2.8)

$$ELR = \log(P[\neg H]) - \log(P[H]) + \int_{-\infty}^{+\infty} \log\left(\frac{\sum_{0\le i\le Mj} P[vj|fj,i]*P[fj,i]}{\sum_{0\le i\le Nj} P[vj|gj,i]*P[gj,i]}\right) * \text{density}(X)dX.$$

The logarithms of the sums could be expanded into Taylor series in order to integrate this expression symbolically, but it is **simpler** to use a numerical integration technique to approximate the value for a specific operator. High-precision values are not needed because they are only used to rank the operators and predict **the** expected number 'of operators required to achieve a certain confidence in I-I.

It is not necessary to integrate the function from minus infinity to plus infinity. Recall the discussion in section 4.3 about "filtering" 'out ***unusual*** values for an operator. Any value that is not within three standard deviations of at least one of the alternatives' means is so unusual that it is treated as a mistake. It is **therefore** sufficient to integrate the function **over** the interval of ***usual*** (or *useful)* values. This interval is simply the union of all the alternatives' intervals defined by their means plus or minus three standard deviations. The resulting interval is finite, which makes it easier to compute the integral numerically.

The result of this section is a set of formulas, which compute the expected contribution of an operator, even if it may involve several known alternatives and surprises. These

expected contributions will be used in later sections to compute other important quantities.

Section 3
COST INFORMATION

Since different operators cost different amounts to apply, a slightly more sophisticated ranking scheme can rank the operators according to a cost-adjusted version of their expected contribution, ie.

$$\frac{\text{<expected log-ratio>}}{\text{<expected cost>}} \qquad (6.3.1)$$

The cost of applying an operator could involve such factors as training time, computation time and memory space, but in this discussion, for simplicity the expected cost of an operator is defined to be the expected computation time required to locate a *match.*

Computation time is a function of several variables: (1) the initialization time, (2) the number of times the operator is applied, and (3) the computation time for each application. If an operator is applied over a complete region (eg. the tolerance region about some alternative), it is relatively easy to predict the expected cost. However, if an operator is sequentially applied in a region (using some search strategy) until a *reasonably good* match is found, one has to predict the number of separate applications -to be used to find such a match. This prediction is a little more difficult. It is based upon the type of feature, the expected distributions of the feature and its alternatives, and the local characteristics of the operator (eg. the size of the region covered by one application). Each feature-operator-strategy triple needs a separate mechanism for predicting the average number of applications required to find a match. Some of these prediction methods are discussed in a later chapter.

An operator-ranking scheme that incorporates cost estimates is: compute the benefit-cost ratios (as in formula 6.3.1) for each of the operators and order them according to the largest first.

Section 4
LEAST-SQUARES CULLING

As mentioned in section 4.5 the least-squares culling routine requires a minimum number of matches. Let M represent this minimum number. Let N be the number of operators that must be applied in order to find M matches. Since an operator may or may not locate a known alternative (ie. a match), N is greater than or equal to M. This section develops a method for predicting N, given M and an ordered list of operators. The following sections continue to derive methods to compute estimates for the *expected number of operators* required to achieve some goal. It should be pointed out that it is possible to compute an estimate for such numbers by simply applying the operators to enough training pictures and averaging the number of operators needed to reach the desired goal. Often this direct way is the best way to proceed. However, sometimes it is useful to be able to produce an independent estimate of the expected number. The following sections discuss some alternative ways of computing the desired estimates.

In order to predict the average number of operators needed to locate M matches it is necessary to compute each operator's expected contribution toward M. Consider figure 6.4.1. Figure 6.4.1.a shows the possible matches associated with a typical operator: three known alternatives and a surprise (f1,f2,f3, and S). Assume that the *a priori* probabilities for these possibilities are:

(6.4.1)
$$P[f1] = .5,$$
$$P[f2] = .2,$$
$$P[f3] = .1,$$
$$\text{and} \quad P[S] = .2.$$

Figure 6.4.1.b shows the densities associated with the various possibilities, but they are scaled by their *a priori* probabilities of occurring. Figure 6.4.1.c shows the weighted density function for the Operator. That is,

(6.4.2)
$$\text{density}(X) = P[S]*\text{density}(S) + \sum_{j=1}^{3}(P[fj]*\text{density}(fj)).$$

Given a specific value for the operator, the best alternative is the alternative with the highest probability of being the correct match, ie.

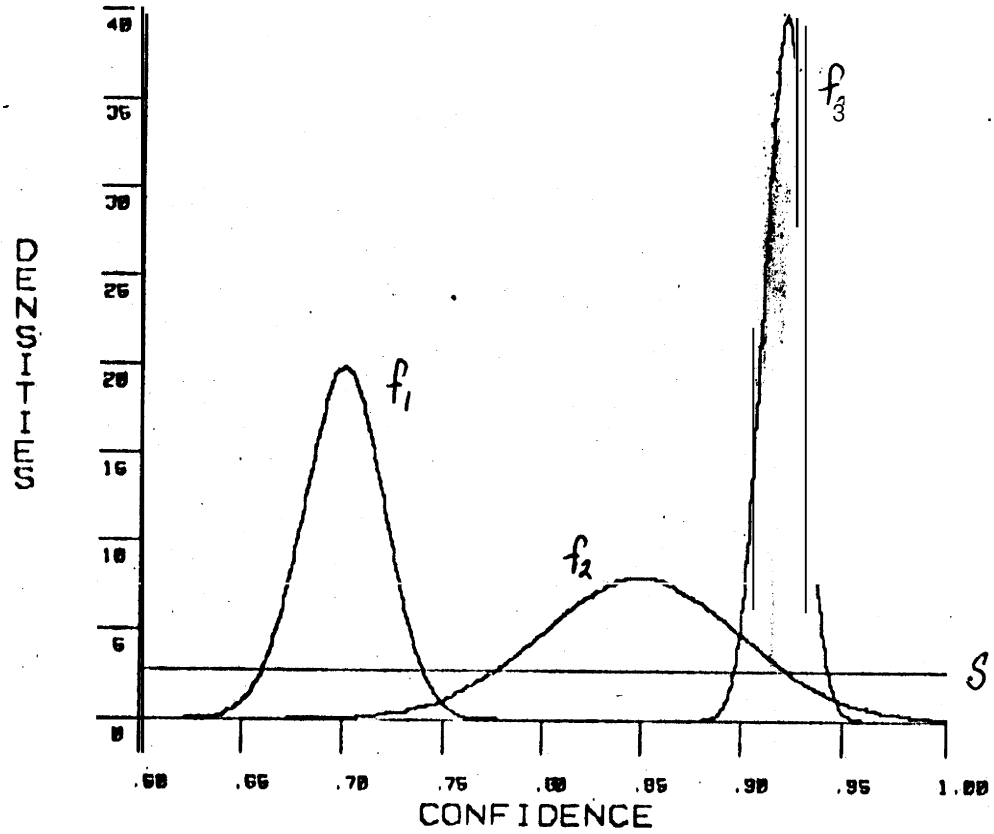


FIGURE 6.4.1.a

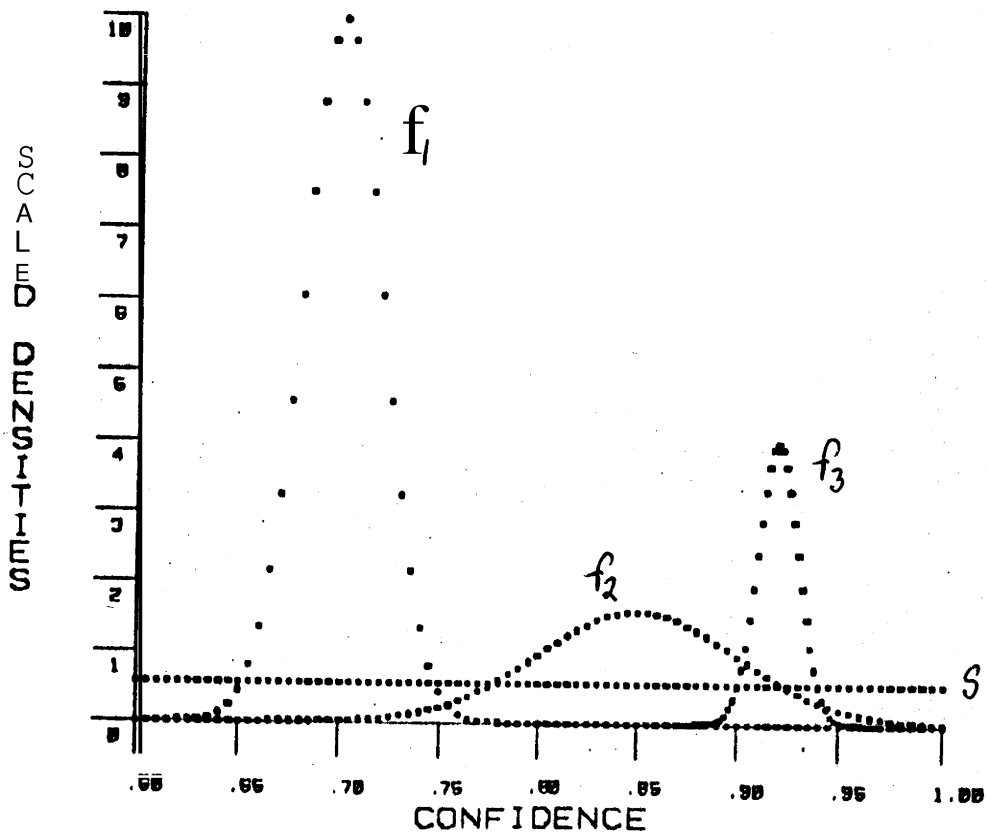


FIGURE 6.4.1.b

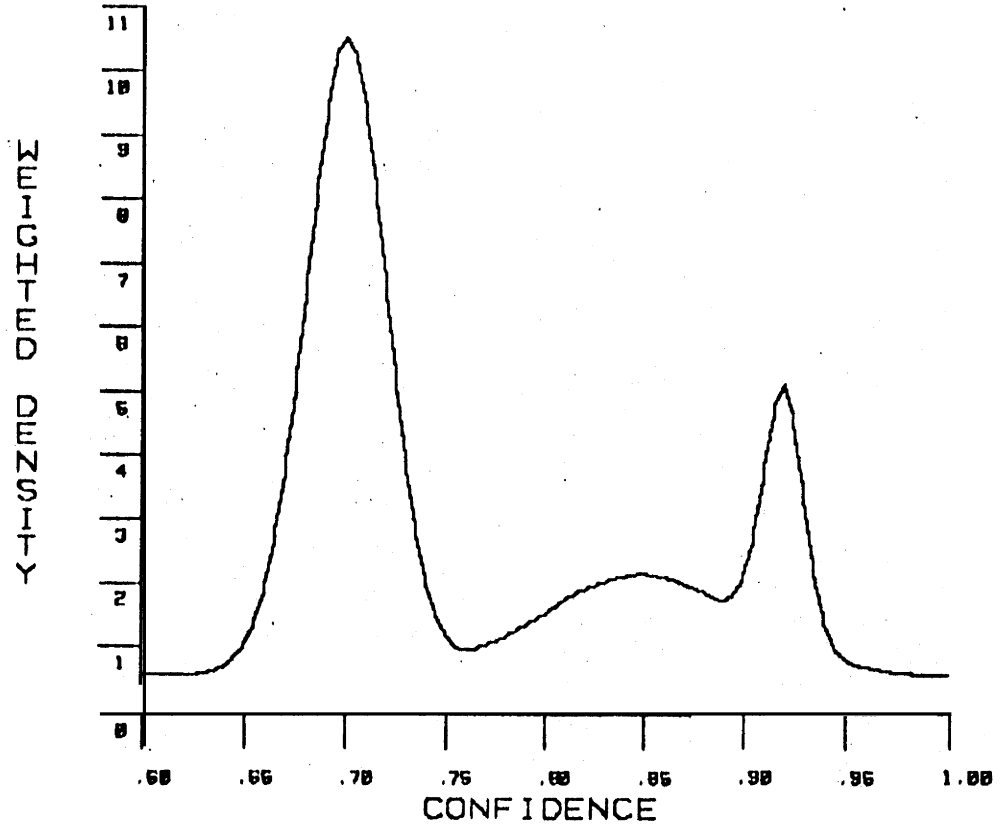


FIGURE 6.4.1.c

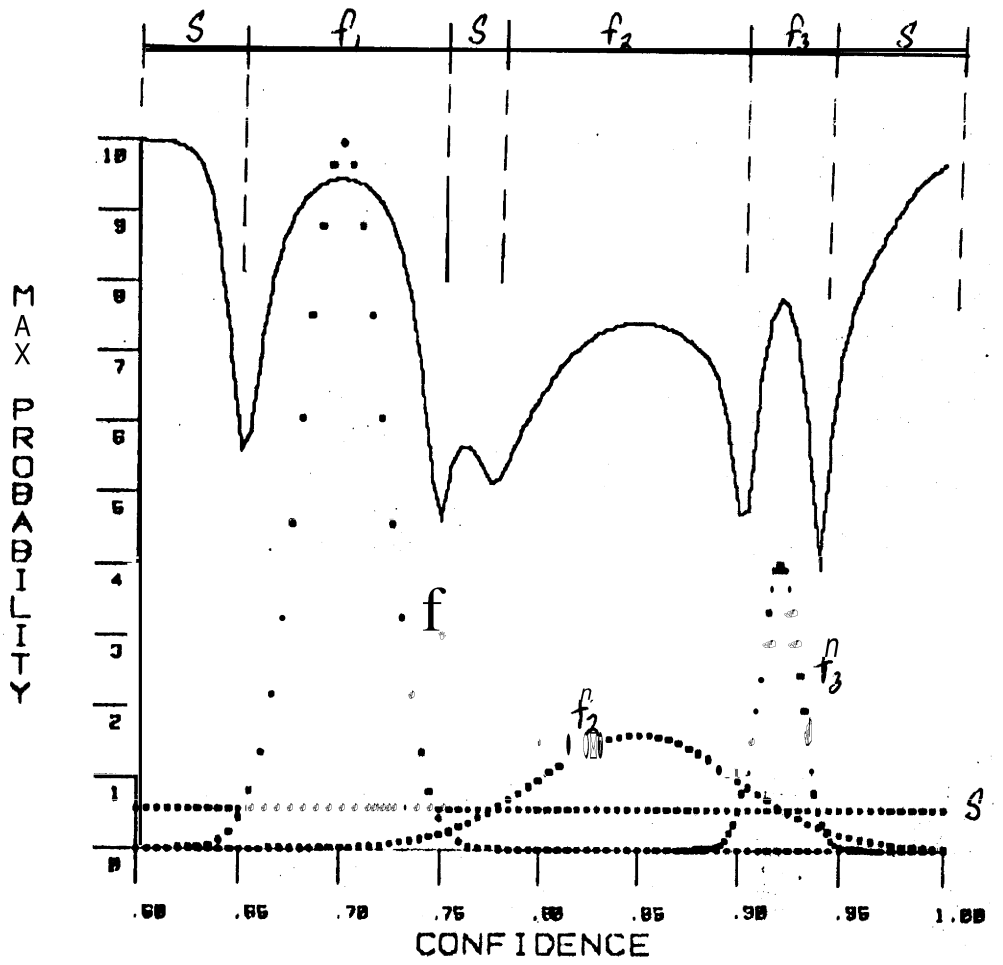


FIGURE 6.4.1.d

(6.4.3) MAX(P[f1|v], P[f2|v], P[f3|v], P[S|v]).

The algorithm shown in figure 4.5.2 uses the probability associated with the best match as the operator's contribution toward the goal of M matches, except when the operator's value is *unusual* or it suggests that a surprise is the best match. In the case of an unusual value or a surprise match, no contribution is credited to the operator. Figure 6.4.1.d superimposes the graph of the operator's contribution (scaled by 10) on top of the scaled densities shown in figure 6.4.i.b. Figure 6.4.1.d also labels each interval with the name of the possibility that would be returned as the best match. Notice that there are three intervals that imply that the surprise is the best match.

The expected contribution of an operator toward M (abbreviated EC) can be computed in the standard way:

$$(6.4.4) \qquad EC = \int_{-\infty}^{+\infty} \langle \text{contribution at } X \rangle * \text{density}(X)\, dX$$

where

$$(6.4.5) \qquad \langle \text{contribution at } X \rangle = \begin{cases} 0 & \text{(if unusual or surprise)} \\ MAX(P[f1|X], \ldots P[fn|X]) & \text{(otherwise).} \end{cases}$$

Again a numerical integration technique is the easiest way of computing the value of EC.

Formula 6.4.4 is important because it computes the expected contribution of an operator. Given an ordered list of operators and their expected contributions it is possible to estimate the number of operators that have to be applied in order to locate M matches. The expected number of operators is the minimum N such that

$$(6.4.6) \qquad \sum_{j=1}^{N} \langle \text{operator } j\text{'s expected contribution} \rangle \geq M.$$

Section 5
INSPECTION

In an inspection task each operator contributes a certain amount toward increasing (or decreasing) the overall confidence that H is true. Sections 6.1 and 6.2 developed methods for computing the *expected* contribution of an operator. Given the expected log-ratio (the contribution) of each operator, what is the expected number of operators required to generate a certain confidence in H? The answer to this question is based upon a theorem in sequential pattern recognition [PR book]:

THEOREM Let e(H) be the error rate allowed for saying that H is true when it really is false and let e(¬H) be the error rate allowed for incorrectly saying that H is false when it really is true. Let

$$A = \frac{1 - e(\neg H)}{e(H)} \qquad \text{and} \qquad B = \frac{e(\neg H)}{1 - e(H)}.$$

Then, given that H is true, the expected number of operators to be used to make a decision is given by

$$\text{expected-\#}(H) = \frac{(1-e(\neg H))*\log(A) + e(\neg H)*\log(B)}{\langle \text{average log-ratio, given } H \rangle}.$$

And given that ¬H is true, the expected number of operators to be used to make a decision is given by

$$\text{expected\_\#}(\neg H) = \frac{e(\neg H)*\log(A) + (1-e(H))*\log(B)}{\langle \text{average log-ratio, given } \neg H \rangle}.$$

And finally, the expected number of operators to achieve the specified error rates is

$$\text{expected-\#} = P(H)*\text{expected\_\#}(H) + P(\neg H)*\text{expected\_\#}(\neg H).$$

1. 2.9
2. 2.1
3. 2.0
4. 1.7
5. 1.6
6. 1.4
7. 1.2
8. 1.2

Expected log-likelihood ratio

FIGURE 6.5.1

The theorem is based upon the assumption that there are an infinite number of operators whose average log-ratios are known. However, there are only a finite number of operators (usually on the order of ten) for any specific VV task. The theorem can still be used to produce an approximate number of operators expected by assuming that there are an infinite number of operators with the contribution of the best operator. If that were the case, how many operators would be needed? If the answer is one or less, then the best operator will probably be sufficient, on the average. If the answer is more than one, consider the average of the first two operators and compute the number needed if there were an infinite number of operators with that expected ratio. If the answer is less than or equal to two, the best two operators will be enough on the average, etc. Figure 6.5.1 lists eight operators and their expected log ratios. Using those operators and a goal of e(H) = e(¬H) = .05, the expected number of operators would be one. The expected number of operators to achieve e(H) = e(¬H) = .005 would be three.

This theorem is powerful because it provides a way of predicting the number of features, on the average, that will be necessary to achieve a specific confidence. The theorem applies to all operators whether or not they have several known alternatives and/or surprises, The only effect of alternatives and surprises is that the operator's expected contribution will probably be smaller than if it did not have such potential confusions.

Section 6
PRECISION

Chapter 5 developed a method to locate objects (in the domain of VV). The method divided a location task into three subtasks:

(1) locate enough features to be able to apply the least-squares culling routine (this set of features is referred to as the kernel),

(2) locate enough additional features to produce the desired precision about the point of interest,

and (3) locate enough additional features to develop the required amount of confidence in the statement of precision.

In order to predict the total number of operators needed in a location task, one needs estimates for each of the subtasks. Section 6.4 developed a method to predict the expected number of operators required in subtask (1). This section and the next section will develop

methods to predict the expected number of operators required by subtasks (2) and (3), respectively.

Given an edge operator and a specific line to be found, the edge operator will be able to locate a point on the line to within some precision. Given a different line (maybe a fuzzy line), the precision of the same edge operator will probably be different. Thus, there is a precision associated with each operator-feature pair. In fact, the precision of most operators also depends upon the *type* and *amount* of change between the planning picture and the actual picture (eg. the amount of rotation or the change in the overall light level). In order to predict the number of operators needed it is necessary' to have a model of each operator's precision. A statistical model provides the variance about each value. Given the variances about the operators' values, the weight matrix (ie. W) can be constructed, which makes it possible for the least-squares routine to determine the variances about the resulting parameter values.

In VV, and in particular in programmable assembly, one assumes that there are no large unknown changes between the planning picture and the actual picture. The environment is highly constrained. The main factors that affect an operator's precision are (1) the inherent operator characteristics (eg. its maximum resolution), (2) the local feature characteristics (eg. fuzziness), and (3) small rotations (eg. 15 degrees). Often the operator's inherent characteristics are the dominant factors involved in determining an operator's precision. In this case an *a priori* estimate can be used to mociel the precision. If this is not true, it is possible to apply the operator (in conjunction with several other operators with known precisions) to several trial pictures and produce an estimate for the operator's variance.

One property'of the least-squares fitting technique is that it produces essentially the same precision no matter what 'the position values from N matches are, as long as they conform to the stated variances. Therefore the precision produced by any one application of the fitting routine can be used as an estimate for the precision from the N operators. This property is the basis of a straightforward method that predicts the number of features needed to reach a certain precision: Given a trial picture, locate a kernel set of matches, and apply the least-squares technique. If the resulting precision is sufficient, stop and return the number of operators used as the expected number operators to be needed. If the precision is not sufficient, locate another match, apply the least-squares routine, and repeat the precision check.

Section 7
CONFIDENCE IN A PRECISION

As mentioned in section 5.2 it is often reasonable to assume that the confidence in the precision is high enough whenever the least-squares routine produces the desired precision. Under this assumption, the expected number of operators required for a location task is the same as the expected number of operators needed to reach the desired precision. If this assumption is not true it is possible to use a method similar to the one used in section 6.5 to estimate the expected number of operators required to reach a certain confidence level.

Formula 5.3.11 shows each operator's contribution toward the overall confidence. Given this symbolic expression for the contribution, it is possible to employ a numerical integration routine to compute the *expected* contribution from an operator. The sequential pattern recognition theorem referred to in section 6.5 can be applied again. Given the expected contributions for the individual operators, the theorem produces the expected number of operators to be needed.

The general prediction scheme for location tasks can now be stated: determine the expected number of operatdrr required to achieve the desired precision, determine the expected number of operators required to reach the desired confidence, and return the maximum of these values as the expected number required for the task.

Section 8
EXPECTED COST

Given (1) an ordered list of operators and (2) the expected number of operators (ie. N) required to achieve a certain goal (either an inspection or location goal), it is easy to produce an estimate for the expected cost associated with achieving the goal: sum the expected costs for the first N operators. That is

(G.8.1) $$\sum_{j=1}^{N} \text{<expected cost of operator j>}.$$

This expression is just a rough estimate for the expected cost because it assumes that the expected cost is the sum of the expected costs for the expected number of operators, which is not generally true. A better estimate is:

$$\sum_{j=0}^{\infty} (P[Aj] * Cj), \tag{6.82}$$

where **Aj** means that the goal is achieved after applying operators one through j and **Cj** denotes the expected cost of applying the first **j** operators.

CHAPTER 5

ASSUMPTIONS

This chapter discusses the assumptions that are required by the mathematical formulas used to compute confidences and precisiona These assumptions are fundamental assumptions about the *class* of tasks referred to as verification vision tasks and about the probabilistic and least-squares *methods* used to model such tasks. An example of such an assumption is the conditional independence of the operators' value and position information. If this assumption is not approximately true for a particular task, none of the Bayesian probability formulas can be applied; their preconditions are not satisfied.

The assumptions have been classified into three types: (1) Bayesian probability assumptions, (2) value distribution assumptions, and (3) conditional independence assumptions. Each type will be discussed in a separate section.

Section 1
BAYESIAN PROBABILITIES

Bayes' theorem states a desired *a posterior-i* probability in terms of the a priori and conditional probabilities:

$$P[H|v] = \frac{P[v|H]*P[H]}{P[v|H]*P[H] + P[v|\neg H]*P[\neg H]}. \qquad (7.1.1)$$

This formula is convenient because the conditional probabilites $P[v|H]$ and $P[v|\neg H]$ are generally easier to measure (or estimate) than $P[H|v]$. However, Shortliffe and Buchanan (see [Shortliffe and Buchanan 75] and [Nilsson 751) have pointed out two related problems involved, in applying Bayes' theorem to various decision tasks. The first problem is that it is often difficult to estimate $P[v|\neg H]$, especially if H is a compound proposition (ie. it is a con junction of several propositions). It is often unclear what the negation of H means. The

second problem is that the amount of statistics required to estimate P[V|¬H] can easily become prohibitive, even if it is clear *what* statistic&should be gathered.

The Bayesian formulas developed for VV in chapters 4 through 6 avoid the first problem by insuring that H is not a compound proposition. For example, in inspection tasks H represents the proposition that the object (eg. the screw) is there. The negation of H denotes the proposition that the object is not there. There are no other possibilities. Within programmable assembly this assumption is reasonable because the environment is highly controlled; the screw is either on the end of the screwdriver *or* it is not. The environment is so predictable in programmable assembly that even the objects that form the background (behind the screwdriver and screw) are known in advance.

VV relies heavily upon the assumption that there are only two possible events that can occur. If there are more than two possible events, other techniques have to be used because the "H or ¬H" model is insufficient. Consider the task of deciding whether a carburetor subassembly has been attached or not. The assumption that H or ¬H is true implies that the only two possibiities are: (1) the carburetor is attached and in its proper place and (2) it is not there at all. If a third aiternarive is possible (eg. a carburetor of the wrong type is attached), the VV formulas are not directly applicable. It might be possible to extend the formulas to cover three or four possibilities, but the modified VV techniques would essentially be recognition-type techniques that choose the best match from several possibilities. Some of the power of the specialized VV techniques would be lost.

It should be noted that there is a difference between three or four *known alternatives* for an operator and three or four possible events that can occur. Known alternatives for an operator are local to the operator. Possible events are global and hence affect ail of the operators. In particular, for each possible event there may be several known alternatives for each operator. Therefore, it is quite a different problem to provide for several different events than it is to provide for several known alternatives. The formulas developed for VV deal with the latter, but not the former.

Even though the VV formulas avoid Shortliffe and Buchanan's first problem, they do not avoid the second. Since they provide for several known alternatives for each operator, the training session has to gather statistics for ail of the alternatives. Fortunately, in programmable assembly there is usually no shortage of potential operators so operators with several known alternatives can often be avoided. In theory the ordering criteria for the operators should include a measure of the expected training'time and the space required for the alternatives. These additions would automatically reduce the rating for an operator that has several known alternatives and reduce the chance that the operator would be used in the 'task.

Section 2
VALUE DISTRIBUTIONS

Throughout the development of the formulas a normal distribution was assumed for the operaters' value information. That is, the values associated with an alternative were assumed to have a normal distribution. This assumption, however, is *not* necessary to compute the likelihood ratios

$$\frac{P[v_i|H]}{P[v_i|\neg H]}. \qquad (7.2.i)$$

Any distribution is sufficient. It is even possible to use the histogram of values produced at training time as the distribution, as long as there is a sufficient number of trials.

A normal distribution was assumed in the derivations because it is a good model fot several of the operators. If the values of an operator are not normally distributed, there may he a change of variable that can convert them into an approximately normal distribution. A later portion of this section will discuss a change of variable that converts correlation values into a distribution that is approximately normal.

If an operator's values are known to follow some distribution other than a normal distribution, it is easy to incorporate the new distribution into the execution-time formulas. The only information needed in addition to the density function is a specification for the interval of *reasonable* values. What values of the operator should be classified as *unusual* and hence should be filtered out (see section 4.5)? For normal distributions it is easy to specify an interval by setting a threshold in terms of the number of standard deviations away from the mean. Other distributions require some ocher specification for the interval of reasonable values.

It is a little more difficult to incorporate an operator into the *planning-time* formulas if its values form some distribution other than a normal distribution. It requires a different function to be integrated in order to compute the operator's expected log-likelihood ratio. However, since the integration can be done numerically, the extension to a new distribution generally only requires a straightforward modification of the existing routines.

One of the main points of this section is that any distribution can be used for an operator's value information. For example, if some *a* priori 'information implies that the

distribution for a particular operator is a gamma distribution, a gamma distribution can be substituted into the appropriate formulas. If the training results imply that the distribution is not one of the standard distributions, the density function defined by the histogram can be used in the formulas.

One operator that is known to produce a non-normal distribution is cross-correlation (see [Hoel 71]). Consider the following formula for the correlation coefficient:

$$r = \frac{\sum_{i=1}^{N} (Xi-Mx)*(Yi-My)}{N * Sx * Sy}, \tag{7.2.2}$$

where Xi and Yi are jointly normally distributed, Mx and My are the sample means of X and Y, respectively, and Sx and Sy are the sample standard deviations. It would be possible to use the actual distribution of r, but there is a convenient change of variable that converts r into a distribution that is approximately normal. The change of variable is

$$z = \frac{1}{2} * \log\left(\frac{1 + r}{1 - r}\right). \tag{7.2.3}$$

The mean of the new distribution is

$$Mz = \frac{1}{2} * \log\left(\frac{1 + a}{1 - a}\right), \tag{7.2.4}$$

where α represents the theoretical value of the correlation coefficient. The standard deviation for the new distribution is

$$St = \frac{1}{\text{sqrt}(N - 3)}, \tag{7.2.5}$$

where N is the number of samples used to compute r.

The correlation operator implemented by Hans P. Moravec at Stanford behaves according to this theory. Consider figure 7.2.1. Figure 7.2.1.a is a histogram of fifty correlation coefficient values. The values are the results of applying the same correlation operator to fifty different pictures of a scene for one VV task. The interval site along the

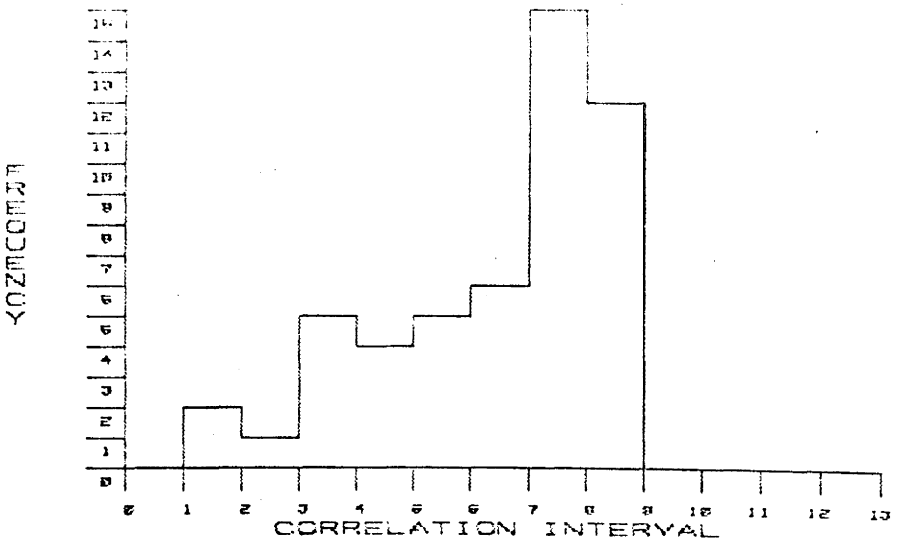


FIGURE 7.2.1.a

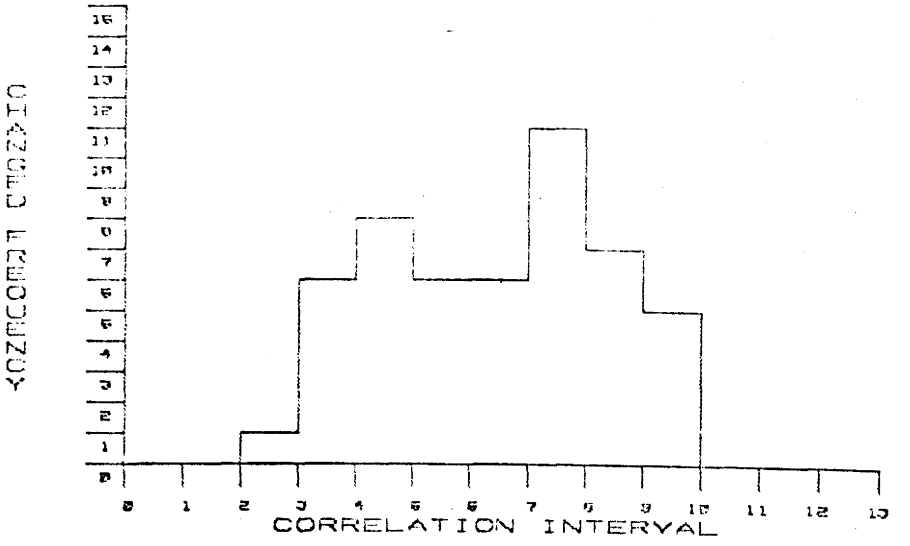


FIGURE 7.2.1.b

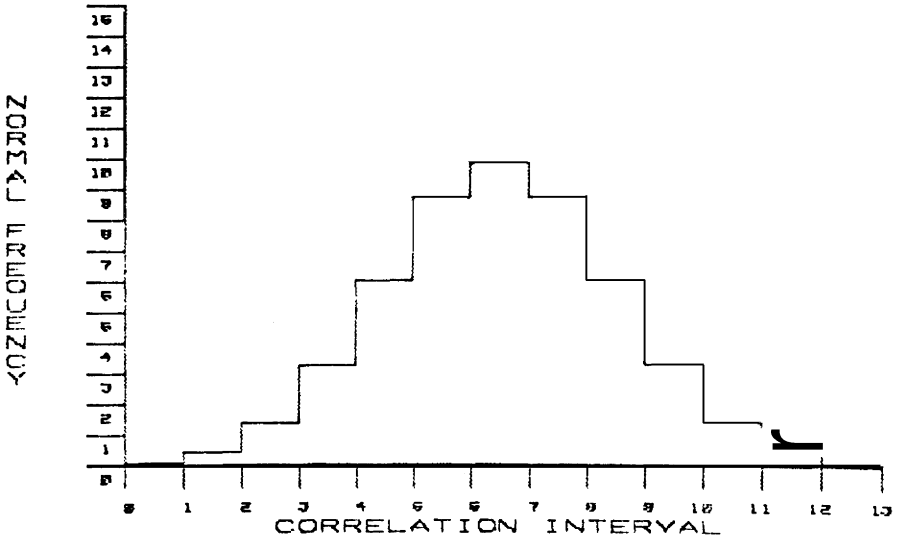


FIGURE 7.2.1.c

horizonal axis of the graph is one-half of the sample standard deviation. As predicted the correlation values form a skewed distribution (with a theoretical upper limit of 1.0). The chi-square value is based upon the eleven intervals centered about the sample mean. Figure 7.2.1.b is the histogram of values produced by the change of variable in formula 7.2.3. Figure 7.2.1.c is the histogram that would be expected if the sample formed a perfect normal distribution.

The chi-square value drops significantly from 25.4 (with eight degrees of freedom) to 9.3 (with eight degrees of freedom) for the new distribution. The improvement is not always that dramatic, but the change of variable seldom increases the chi-square value. Consider figure 7.2.2. It is a scatter diagramof the pairs:

(7.2.6) (<chi-square of raw values), <chi-square of changed values>).

Any point to the right of the diagonal line represents a case in which the change of variable made the distribution for an operator's values look more like a normal distribution (according to the chi--square test). The change of variable only slightly degrades the chi-square value in the few cases that it makes the distribution worse. A point in the shaded area of figure 7.2.2 represents an operator whose distribution was improved significantly. Eefore the change of variable the chi-square test (at the 5% level) rejected the hypothesis that the sample could have come from a normal distribution. After the change of variable the chi-square test indicated that it was plausible for the sample to have come from a normal distribution.

The question 'about which distribution to use to model an operator's results is a hard one. The chi-square test used above is helpful, but mainly as a method for rejecting a proposed model.

After deciding which distribution to use to model an operator's results, one still has to decide how many samples are needed to produce a good approximation to the distribution. If the chosen distribution is normal, one needs enough samples to appoximate the mean and standard deviation, since a normal distribution is completely determined by these two parameters. How many samples are needed? There are two theorems that help answer these questions (see [Hoel 71]):

THEOREM: If X is normally distributed with variance V and

$$VS = \frac{\sum_{i=1}^{N} (Xi - MS)^2}{N}$$

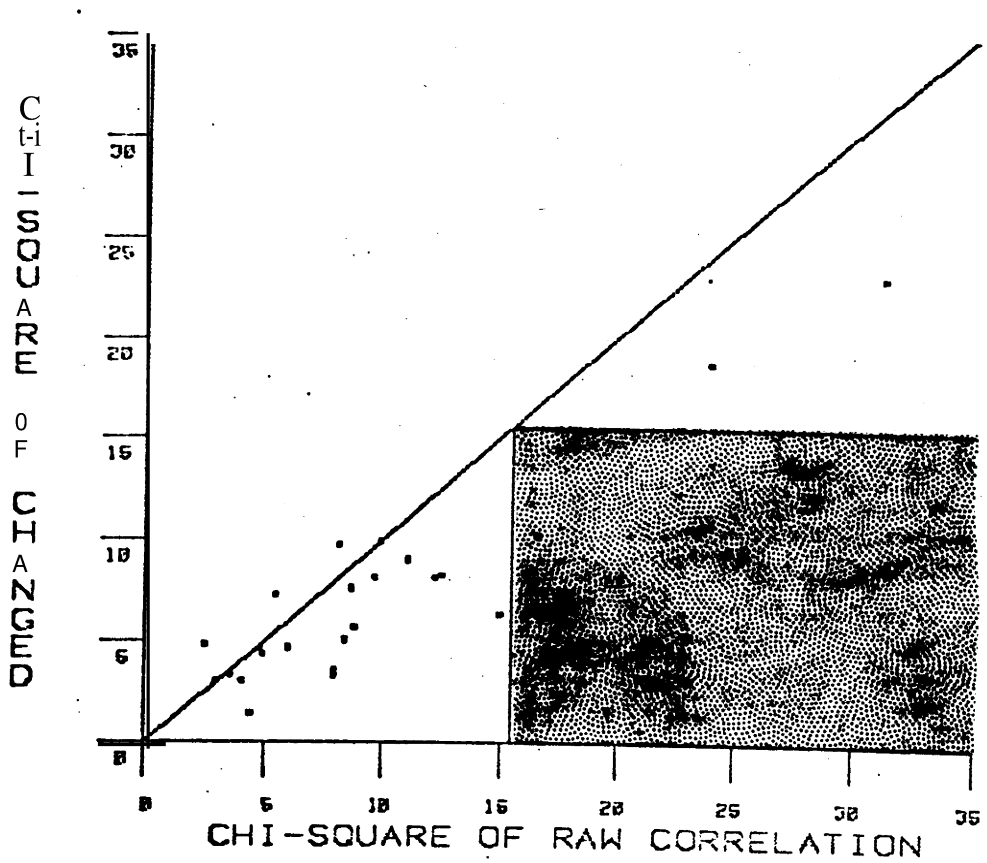

CHI-SQUARE OF CHANGED
0
5
10
15
20
25
30
35
CHI-SQUARE OF RAW CORRELATION

FIGURE 7.2.2

is the sample variance based upon a random sample of size N and Ms is the sample mean, then

$$\frac{N*Vs}{V}$$

has a chi-square distribution with (N-l) degrees of freedom.

THEOREM: If X is normally distributed with mean M and variance V and a random sample of size N is taken, then the sample mean Ms will be normally distibuted with mean M and variance V/N,

Let CS(n,p) represent the value such that a chi-square distribution with n degrees of freedom has p percent of the population to the right of that value. One application of the first theorem states that there is a ninty-five chance that the sample variance and actual variance are related as. follows:

$$(7.2.7) \qquad CS((N-1),.975) \le \frac{N*Vs}{V} \le CS((N-1),.025).$$

Let S and Ss represent the standard deviation and the sample standard deviation of the distribution. Since Vs = Ss*Ss and V = S*S, formula 7.2.7 can be converted into the following statement concerning the actual and sample standard deviations:

$$(7.2.8) \qquad \frac{sqrt(N)*Ss}{sqrt(CS((N-1),.025))} \le s \le \frac{sqrt(N)*Ss}{sqrt(CS((N-1),.975))}.$$

The second theorem can be used to produce a ninty-five percent confidence interval about the mean. That is,

$$('7.2.9) \qquad |M - Ms| \le \frac{2*S}{sqrt(N)}$$

or, substituting the larger value from (7.2.8) into (7.2.9) produces

$$(7.2.10) \qquad |M - Ms| \le \frac{2*Ss}{sqrt(CS((N-1),.975))}.$$

For example, if Ms = 1.3 and Ss = .2, the ninty-five percent confidence intervals based upon 8 sample size of 15 are

(7.2.11) $$|M - Ms| \leq .16950$$
$$\text{and } .15092 \leq S \leq .32824.$$

For a sample size of 30 the intervals are

(7.2.12) $$|M - Ms| \leq .10022$$
$$\text{and } .16151 \leq S \leq .27446.$$

One interesting possibility is to use the planning-time formulas to predict the effect of gathering more samples from an operator's distribution. Two important questions can be answered in this way:

(1) Given a sample mean and a sample standard deviation, plus confidence intervals about them, what is a reasonable, but *conservative* distribution (or set of distributions) that can be used to model the operator?

(2) Given an additional set of N samples from a distribution, what is the probable change in the operator's expected contribution?

In this situation a *conservative* distribution is a distribution that understates the contribution of the operator. The use of such a distribution may require more operators to be. applied than theoretically necessary, but there is a smaller chance of making an incorrect decision. For example, assume that a potential operator in an inspection task has the following characteristics:

(7.2.13)

(sample size of 15)	
H	¬H
Ms = 1.3	Ms = 1.95
SS = .2	SS = .22

Assume that the probability of H is .9. Then the expected log-ratio for the operator is 3.41. 7'0 pick a more conservative distribution for the operator consider the sixty percent confidence intervals about the means and standard deviations:

(7.2.14)

(sample size of 15)	
H	¬H
1.234≤ M ≤1.365	1.8785 M≤2.022
.182≤ S s.252	.200≤ S s. 277

If one assumes that the the most conservative set of distributions is produced at the extremes of these intervals, there are sixteen possible combinations for the pair of distributions to be used to model the operator. Figure 7.2.3 shows the expected contribution of the operator for each of the sixteen possibilities. The most conservative set is the set that has the lowest expected contribution, ie.

(7.2.15)

(sample size of 15)	
H	¬H
M= 1.365	M = 1.878
S =.182	s = .277

(the expected log-ratio is 1.25).

What is the expected gain from gathering another fifteen samples from the operator's distributions? The intervals are:

(7.2.16)

(sample size of 30)	
H	¬H
1.258s M 51.342	1.904s M ≤1.996
.185≤ S s.231	.203≤ Ss. 254

and the most conservative distribution (within the sixty percent intervals) is

(7.2.17)

(sample size of 30)	
H	¬H
M = 1.342	M= 1.904
S =.185	S = .254

(the expected log-ratio is 1.80).

The potential gain is significant in terms of the increase in the expected log-ratio for the conservative set of distributions. More samples would increase the expected log-ratio even further. The upper limit on this log-ratio would be reached when the conservative set of distributions was the same as the sample set. At that point the expected log-ratio for both of them would be 3.41. The number of samples actually used in a VV task depends upon *how* conservative the programmer is, how important execution time is, and how much time can be devoted to training the system. Sample sizes on the order of twenty to fifty have worked well.

In programmable assembly since each VV task is performed repeatably, it is possible to

	Expected Log-ratio	
min M, min S1, min M2, min S2:	4.04	
min M, min S1, min M2, mex S2:	1.90	
min M, min S1, max M2, min S2:	6.03	
min M, min S1, max M2, max S2:	2.79	
min M, max S1, min M2, min S2:	4.38	
min M, max S1, min M2, max S2:	2.11	
min M, max S1, max M2, min S2:	6.52	
min M, max S1, max M2, max S2:	3.15	
max M, min S1, min M2, min S2:	2.57	
max M, min S1, min M2, max S2:	1.25	*
max Ml, min S1, max M2, min S2:	4.20	
max M, min S1, max M2, max SZ:	1.98	
max M, max S1, min M2, min S2:	2.81	
max M, max S1, min M2, max S2:	1.35	
max M, max S1, max M2, min S2:	4.56	
max Ml, mex S1, mex M2, max S2:	2.20	

* the minimum expected log-ratio, the most conservative set of distributions

FIGURE 7.2.3

gather additional samples during production runs. This is important because a larger. set of samples can help to refine the model for an operator in two ways. First, more samples can improve the distributions being used to model the operator, and second, if one of the global variables (eg. lighting or camera sensitivity) changes slowly over time, continuous sampling can maintain an **up-to-date** model for the operator.

Section 3
CONDITIONALINDEPENDENCE

The derivations of several of the formulas depend upon two important assumptions about the conditional independence of the operators' values: (1) the value of an operator is conditionally independent of the values of the other operators and (2) the value of an operator is conditionally independent of the position of its match. Both of these assumptions are instrumental in simplifying the relevant formulas. For example, in chapter 4 they make it possible to simplify formula (4.5.2):

$$(7.3.1)\qquad P[H|v1,\ldots vN,p1,\ldots pN] = \frac{1}{1 + \frac{P[v1,\ldots vN,p1\ldots pN|\neg H]}{P[v1,\ldots vN,p1\ldots pN|H]} * \frac{P[\neg H]}{P[H]}}.$$

into

$$(7.3.2)\qquad P[H|v1,\ldots vN,p1,\ldots pN] = \frac{1}{1 + \prod_{i=1}^{N} \frac{P[vi|\neg H]}{P[vi|H]} * \frac{P[p1\ldots pN|\neg H]}{P[p1\ldots pN|H]} * \frac{P[\neg H]}{P[H]}}.$$

The assumptions significantly reduce the number of dependencies in the conditional probabilities and make them feasible to compute.

-There are several reasons why the appearance of a feature might change from one picture to the next:

(1) The feature itself may be different. For example, in assembly tasks all of the pump bases arc not exactly the same. In

photo-interpretation (abbreviated PI) the roads may be widened, or otherwise changed.

(2) The position and orientation of the objects in the picture may change. In assembly the range of positions and orientations for an object is generally specified in advance by an assembly engineer.

(3) The lighting may be different. The sun may be in a different location, causing different shadows and glares. In programmable assembly the lights can be controlled more easily, but they still may vary slightly.

(4) The position and orientation of the camera may be different. In assembly a camera may be in a fixed location, or it may be calibrated to a certain precision. In PI the inertial guidance system specifies the position and orientation of the camera (to within some uncertainty) when a picture is taken.

(5) The sensitivity of the camera may be different. All cameras have internal parameters such as the target voltage that change over time.

(6) The noise level is variable.

All of these sources of change can be considered to be global variables with respect to a VV task. In effect the two conditional independence assumptions state that in VV none of these variables change the expected distribution of values produced by an operator.

There do exist operators and situations for which the assumptions are not true. This fact raises two important questions: (1) in general are the assumptions true for a sufficient number of operators to accomplish practical VV tasks? and (2) is there a way of determining if the assumptions are true for an operator in a specific situation? The remainder of this section develops some insight into the complexity of these questions by analyzing the assumptions further and investigating some of the situations in which they are not true.

The first assumption states that the value of an operator is conditionally independent of the values of the other operators, eg.

(7'.3.3) $$P[v2|H,v1] = P[v2|H].$$

This formula says that given H, the probability of operator 2 producing the value v2 is the

same whether or not the value of operator 1 is known. This statement is generally true in verification vision. The probability of producing a certain value for a correlation operator is generally unaffected by knowing the results of a a previously applied edge operator. An obvious case in which the assumption is not true is when operator 1 and operator 2 are both correlation operators and they overlap. Knowing the value of one operator would certainly alter the possible values for the second. However, overlapping operators are quite uncommon and can be easily avoided in VV.

The second assumption states that the appearance of a feature on an object does not change as the object moves through its possible positions. Put another way, if an operator is applied to several different pictures, and it locates the same known alternative in each, the value returned by the operator is independent of the location of the alternative in the picture. This is generally true in programmable assembly and VV because the changes are so small that the appearance of a feature is essentially constant.

There are two situations in which the second assumption might be false. The first is when a small change in one of the transform's variables causes a shadow to fall on a feature. At some locations the feature is in a shadow and at others it is not. The value of almost any operator attempting to locate such a feature would depend upon whether the feature is in a shadow or not as determined by the position. Hence the value of the operator depends upon the position of the feature, which makes the assumption false. . The second situation arises when a small change in position causes a dramatic change in the appearance of a feature. As an example one view may show a screw hole that is partially occluded on the left by a shaft and a second view of the same hole may show the shaft occluding the hole on the right. This problem is the standard problem of *degenerate views* first referred to in conjunction with the blocks *world.*

Both of these situations lead to operators that produce bivariate (or at least high variance) density functions. One peak is produced by the pictures showing the feature in the shadow and the other peak is produced by the pictures showing it in the light. Since the expected contribution for such operators is generally low, the automatic ranking scheme will place this type of operator near the bottom of the list of potential operators to be used in a task. If the VV system is interactive, a programmer could discard this type of feature if one were suggested by the system,

CHAPTER 6

BIBLIOGRAPHY

Andrews, H. C. [1972], "Introduction to Mathematical Techniques in Pattern Recognition," John Wiley and Sons, Inc., 1972.

Bolles, R. C. [1975], "Verification Vision within a Programmable Assembly System: An Introductory Discussion/' Stanford Artificial Intelligence Laboratory Memo f275, December 1975.

Forsythe, G. E. and Moler, C. B. [1967], 'Computer Solution of Linear Algebraic Systems," Prentice-Hall, 1967.

Graybill, F. A. [1961], "An Introduction to Linear Statistical Models," Volume I, McGraw-Hill Book Company, 1961.

Hoel, P. G. [1971], "Introduction to Mathematical Statistics,' John Wiley and Sons, Inc., 1971.

Mathlab Group [1974], "MAXSYMA Reference Manual," Massachusetts Institute of Technology, September 1974.

Nilsson, N. 3. [1975], "Artificial Intelligence - Research and Applications," Stanford Research Institute, May 1975.

Shortliffe, E. H. and Buchanan, B. G. [1975], "A Model of Inexact Reasoning in Medicine," Mathematical Biosciences 23, 351-379, 1975.

Sproull, R. [1976], "(something like: Decision Theory within Artificial Intelligence)", Stanford University Ph.D. Dissertation, 1976.

Taylor, R. H. [1976] "The Synthesis of Manipulator Control Programs from Task-level Specifications," Stanford University Ph.D. Dissertation, 1976.

VII. DISCRETE CONTROL OF THE ARM

Michael D. Roderick

Artificial Intelligence Laboratory
Computer Science Department
Stanford University

The author is currently a Technical Staff Member of the Engineering Systems Division of TRW Corporation, 1 Space Park, Redondo Beach, California. At the time this research was performed, he was a graduate student in the Electrical Engineering Department at Stanford University.

I. INTRODUCTION

A. General

The use of computer controlled manipulators for industrial assembly tasks is becoming increasing popular and feasible. Experimental systems are now being developed which combine world modeling and sensory feedback to complete tasks not previously possible with conventional "pick and place" manipulators which move through a preplanned sequence of points. .

As the application of programmable manipulator systems becomes more widespread, the number of manipulators utilized in an assembly task will increase. In some applications, all of the manipulators will be controlled by a single medium sized computer. In others, microprocessors will be dedicated to individual manipulators. In both of these cases, the computer processing time available for control will be at a premium. The sampling rate of the control systems will then become a critical factor that will determine the number of functions that can be controlled.

The purpose of this thesis is to determine the minimum sampling rate needed to effectively operate a manipulator and to pinpoint those factors which have the most predominant effect on the minimum sampling rate. This thesis deals specifically with the Stanford robot arm located at the Stanford Artificial Intelligence Laboratory. Previous analysis of the Stanford arm control system has been performed in the continuous Laplace transform domain. This approach is accurate for high sampling rates, but at low sampling rates, Laplace transforms cannot accurately model discrete digital computations. An analysis was needed that could accurately represent both discrete computations and the continuous arm servo system at low sampling rates.

In this thesis, the arm is modeled using z transforms, which can represent a sampled-data system exactly at each sampling instant. The model is used to determine the effect of inertia and sampling rate varations on the dynamic response of the arm. Recommendations are then made describing methods for reducing the effects of inertia variations and for running the arm at reduced sampling rates.

B. Description of Stanford Arm

The Stanford Arm has six joints plus a gripper consisting of two fingers with microswitches for touch sensors. Each joint of the arm has a potentiometer and tachometer for sensing position and velocity. Joint torque is determined by measuring the joint motor current.

The arm motor drives generate a current proportional to the command signal from the computer, so that motor torque is directly proportional to the computer command signal. Brakes are applied to each joint when the arm is stopped to eliminate the need to servo the arm continuously. The arm's absolute accuracy is ±0.1 inches and its repeatability is k0.03

inches.

An extensive manipulator programming system, known as "AL", has been developed at the Artificial Intelligence Laboratory for running the arm. The AL system contains a compiler for planning the arm's trajectories and a run-time system which executes the programs generated by the compiler. The compiler is written in a language similar to ALGOL and resides on a PDP-10. The run-time system is written in PALX assembly language and runs on a **PDP-11/45.**

C. Arm Trajectory Calculations

The' arm's trajectory is determined by evaluating the fifth order polynomial given in equation (1- 1) below.

$$\theta_c(k) = a, + a_1[kT] + a_2[kT]^2 + a_3[kT]^3 + a_4[kT]^4 + a_5[kT]^5 \qquad (1\text{-}1)$$

where

$\theta_c(k)$ = command joint position
T = sampling period
k = discrete time variable
ao to a_5 = polynomial coefficients

The command arm velocity, $\omega_c(k)$, is determined by differencing the current and previous positions and then dividing by the **sampling** period.

$$\omega_c(k) = \frac{\theta_c(k) - \theta(k\text{-}1)}{T} \qquad (1\text{-}2)$$

The arm's command acceleration,, $\alpha_c(k)$, is determined by differencing the current and previous velocities and then dividing'by the sampling period.

$$\alpha_c(k) = \frac{\omega(k) - \omega(k\text{-}1)}{T}$$

$$= \frac{\theta(k) - 2\theta(k\text{-}1) + \theta(k\text{-}2)}{T^2} \qquad (1\text{-}3)$$

The z transforms of $\omega_c(k)$ and $\alpha_c(k)$ are given by

$$\omega_c(z) = \frac{(z-1)}{Tz}\,\theta(z) \tag{1-4}$$

$$\alpha_c(z) = \frac{(z-1)^2}{(Tz)^2}\,\theta(z) \tag{1-5}$$

where z = z transform operator.

II. CLASSICAL ANALYSIS OF THE ARM

A. Joint Mode.1

The motor torque difference equation used to control each of the arm's six joints is given by

$$T(k) = J(k)\alpha_c(k) + G(k) + F - k_c k_p [\theta_a(k) - \theta_c(k)]$$

$$- k_c k_v J(k) [\omega_a(k) - \omega_c(k)] - kcki \sum_{j-1}^{k} [\theta_a(j) - \theta_c(j)] / J(j) \qquad (2\text{-}1)$$

where

T(k) = command motor torque (oz-in)
$\theta_a(k)$ = actual joint position (deg)
$\theta_c(k)$ = command joint position (deg)
$\omega_a(k)$ = actual velocity (deg/sec)
$\omega_c(k)$ = command joint velocity (deg/sec)
$\alpha_c(k)$ = command joint acceleration (deg/sec^2)
J(k) = joint inertia ($oz\text{-}in\text{-}sec^2$)
G(k) = joint gravity loading (oz-in)
F = joint friction with same sign' as velocity (oz-in)
k_p = proportional feedback constant (oz-in)
k_v = derivative feedback constant (1/sec)
ki = integral feedback- constant ($oz^2\text{-}in^2\text{-}sec^2$)
k_c = constant to convert degrees to radians = .01745 (rad/deg)

The gravity loading, G(k), is calculated 'using a first order polynomial.

$$G(k) = g_o + \delta_g \, kT/S \qquad (2\text{-}2)$$

where

g_o = initial gravity loading
δ_g = change in gravity loading (oz-in)
T = sampling period (sec)
kT = elapsed time in segment (sec)
S = total time required to pass through segment (sec)
kT/S = fraction of time through segment (0.040)

The inertia, J(k), is also calculated using a first order polynomial.

$$J(k) = j_o + \delta_j \, kT/S \qquad (2\text{-}3)$$

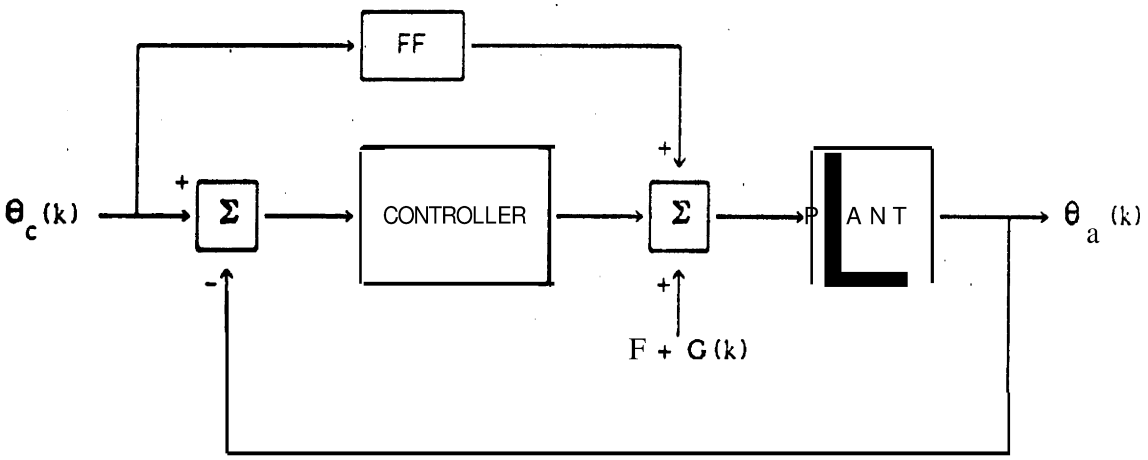
where

j_o = initial inertia (oz-in-set^2)

δ_j = change in inertia (oz-in-sec^2)

kT/S = fraction of time through segment (0.0→ 1.0)

A block diagram of the control system described by equation (2-1) is shown in Figure 2-1. A more detailed diagram of this model is given in Figure 2-2.



where

$\theta_c(k)$ = command arm position (deg)

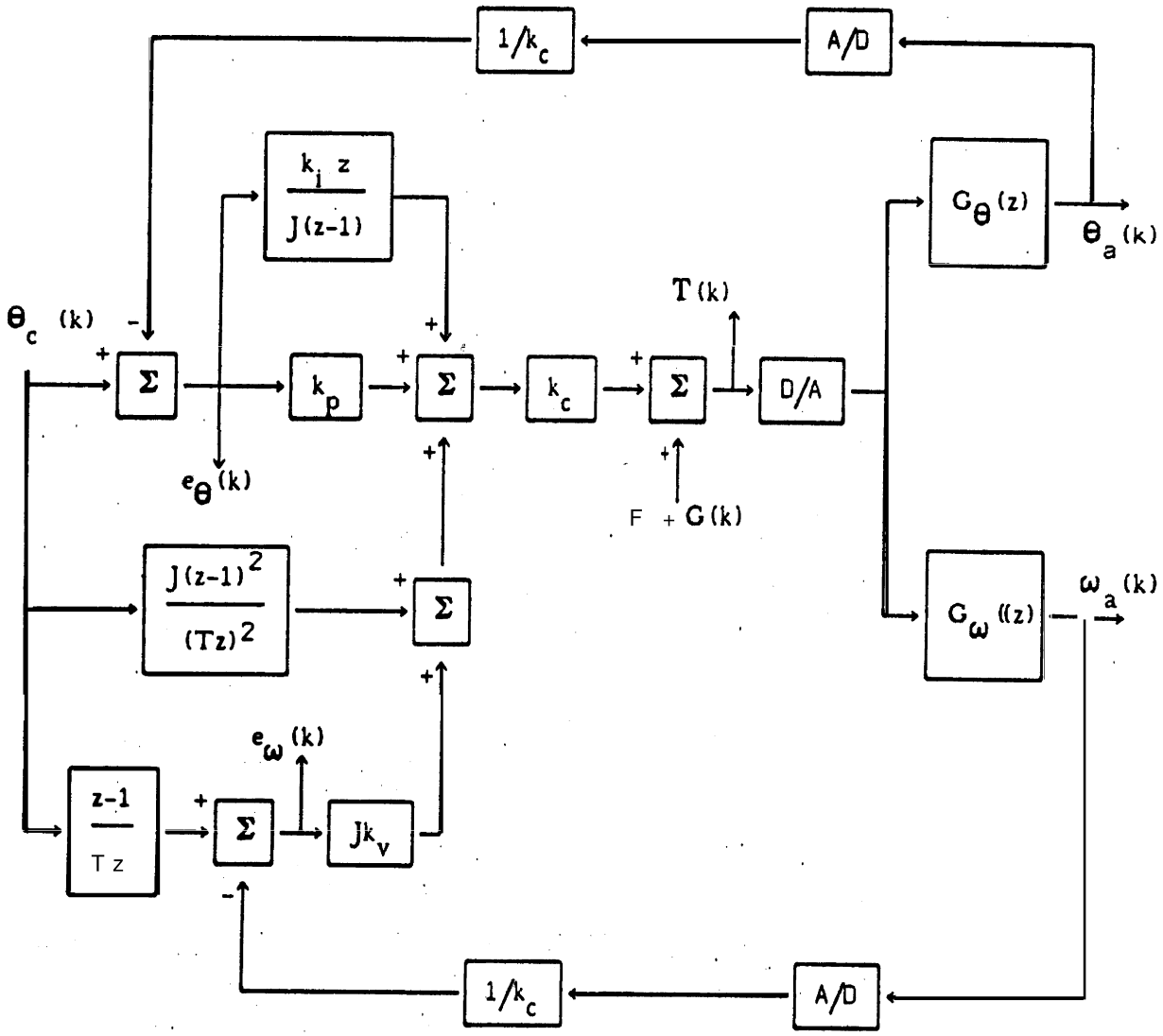
$\theta_a(k)$ = actual arm position (rad)

FF = feedforward compensation for inertia = $J(k)\alpha_c(k)$ (oz-in)

PLANT = one joint of the arm and its associated gear train and motor

CONTROLLER = joint feedback equations

Figure 2-1: joint Model



where

$G_\theta(z)$ = position transfer function of plant (rad/oz-in)
$G_\omega(z)$ = velocity transfer function of plant (rad/oz-in-sec)
A/D = analog to digital converter
D/A = digital to analog converter

Figure 2-2: Detailed Joint Model

B. Joint Transfer Function

The transient response of a servomotor system is described by the differential equation given in equation (2-4).

$$T(t) = J \frac{d^2\theta_a(t)}{dt} + B \frac{d\theta_a(t)}{dt} + F \qquad (2\text{-}4)$$

where

J = Joint inertia (oz-in-set^2)
B = viscous damping constant (oz-in-.set)
F = coulomb friction constant (oz-in)

The inertia in the above equation is assumed to be a constant, although on some of the joints, it is a time varying function which depends on the 'configuration of the arm. The effect of variations in joint inertia will be discussed in Section V-B.

A compensation is made for coulomb friction in the arm control system given in Figure 2-2. A constant offset is added to the motor torque whenever the velocity of the motor is greater than zero. The sign of the offset is the same as the sign of the velocity. With this friction compensation, the effects of coulomb friction are minimized and the friction constant in equation (2-4) can be assumed to zero.

The Laplace transform transfer functions for position and velocity of the arm are then given by

$$G_\theta(s) = \frac{\theta_a(s)}{T(s)} = \frac{1}{Js^2 + Bs} \qquad (2\text{-}5)$$

$$G_\omega(s) = \frac{\omega_a(s)}{T(s)} = \frac{1}{Js + B} \qquad (2\text{-}6)$$

where s = Laplace transform operator.

When a D/A converter is placed in series with an analog 'plant, the response of the analog. plant can no longer be described exactly by continuous Laplace transform transfer functions. A design tool which becomes very useful at this point is the zero-order-hold (ZOH) approximation. The ZOH approximation can model the response of a D/A converter and a continuous plant exactly at each sampling instant. The ZOH approximation is given by

$$ZOH(z) = \frac{(z-1)}{z} Z\left\{\frac{G(s)}{s}\right\} \qquad (2\text{-}7)$$

where Z = z transform operation.

Since each, joint of the Stanford arm uses a D/A converter between the output of the computer and the input of the motor drive, the two transfer functions of the arm system can be derived using the ZOH approximation:

$$G_\theta(z) = \frac{\theta_a(z)}{T(z)} = \left(\frac{z-1}{z}\right) Z\left\{\frac{G_\theta(s)}{s}\right\} = \left(\frac{z-1}{z}\right) Z\left\{\frac{1}{s(Js^2+Bs)}\right\}$$

$$= \frac{(TB + J(\beta-1))z - TB\beta - J(\beta-1)}{B^2(z-1)(z-\beta)} \qquad (2\text{-}8)$$

$$G_\omega(z) = \frac{\omega_a(z)}{T(z)} = \left(\frac{z-1}{z}\right) Z\left\{\frac{G_\omega(s)}{s}\right\} = \left(\frac{z-1}{z}\right) Z\left\{\frac{1}{Js^2+Bs}\right\}$$

$$= \frac{1-\beta}{B(z-\beta)} \qquad (2\text{-}9)$$

where $\beta = e^{-BT/J}$.

The equations for the command torque $T(z)$ have been computed using Figure 2-2 and are given by

$$T(z) = k_c \left[Jk_v e_\omega(z) + \left(k_p + \frac{k_i z}{J(z-1)}\right) e_\theta(z) + \frac{J(z-1)^2}{(Tz)^2}\theta_c(z) \right] \qquad (2\text{-}10)$$

where

$$e_\omega(z) = \frac{(z-1)}{Tz}\theta_c(z) - \frac{G_\omega(z)}{k_c}T(z) \qquad (2\text{-}11)$$

$$e_\theta(z) = \theta_c(z) - \frac{G_\theta(z)}{k_c}T(z) \qquad (2\text{-}12)$$

Combining equations (2-10) through (2-12) gives the transfer function for the controller $G_c(z)$.

$$G_c(z) = \frac{T(z)}{\theta_c(z)} = \{ \frac{k_g k_c B^2 (z-1)(z-\beta)}{(Tz)^2} \} \times \{ \frac{z^3 + d_1 z^2 + d_2 z + d_3}{z^3 + c_1 z^2 + c_2 z + c_3} \} \qquad (2\text{-}13)$$

where

k_g = gain constant = k_{g1}/k_{g2}

$k_{g1} = T^2 [k_i + k_p J] + J^2 [k_v T + 1]$

$k_{g2} = JB^2$

$b_1 = TB + J(\beta - 1)$

$b_2 = TB\beta + J(\beta - 1)$

$c_1 = \{ b_1 [k_i + k_p J] - JB [B(2 + \beta) + k_v J(\beta - 1)] \}/k_{g2}$

$c_2 = \{ JB [B + 2B\beta + 2k_v J(\beta - 1)] - b_2 [k_i + k_p J] - k_p J b_1 \}/k_{g2}$

$c_3 = \{ k_p J b_2 - JB [B\beta + k_v J(\beta - 1)] \}/k_{g2}$

$d_1 = \{ -J [k_p T^2 + J [2k_v T + 3]] \}/k_{g1}$

$d_2 = \{ J^2 [k_v T + 3] \}/k_{g1}$

$d_3 = \{ -J^2 \}/k_{g1}$

Combining equations (2-8) and (2-13) gives the closed loop transfer function from $\theta_c(z)$ to $\theta_a(z)$:

$$H(z) = \frac{\theta_a(z)}{\theta_c(z)} = \frac{k_c k_g [b_1 z - b_2] [z^3 + d_1 z^2 + d_2 z + d_3]}{(Tz)^2 [z^3 + c_1 z^2 + c_2 z + c_3]} \qquad (2\text{-}14)$$

There are several simplifications of the above transfer function that will be used later in other sections. The first of these is the transfer function of the arm without the feedforward coinpensation term, $J\alpha_c(k)$:

$$H(z) = \frac{\theta_a(z)}{\theta_c(z)} = \frac{k_c k_g [b_1 z - b_2] [z^2 + e_1 z + e_2]}{Tz [z^3 + c_1 z^2 + c_2 z + c_3]} \qquad (2\text{-}15)$$

where

k_g = gain constant = k_{g3}/k_{g4}

$kg3 = T[k_i + k_p J] + k_v J^2$

$k_{g4} = JB^2$

$b_1 = TB + J(\beta - 1)$

$b_2 = TB\beta + J(\beta - 1)$

$c_1 = \{ b_1 [k_i + k_p J] - JB [B(2 + \beta) + k_v J(\beta - 1)] \}/k_{g4}$

$c_2 = \{ JB [B + 2B\beta + 2k_v J(\beta - 1)] - b_2 [k_i + k_p J] - k_p J b_1 \}/k_{g4}$

$c_3 = \{ k_p J b_2 - JB [B\beta + k_v J(\beta - 1)] \}/k_{g4}$

$e_1 = \{ -J [k_p T + 2k_v J] \}/k_{g3}$

$e_2 = \{ k_v J^2 \}/k_{g3}$

The second simplified tranifer function includes proportional and derivative feedback only. The feedforward and integral feedback terms have been deleted.

$$H(z) = \frac{\Theta_a(z)}{\Theta_c(z)} = \frac{k_c k_g [b_1 z - b_2][z + g]}{Tz[z^2 + f_1 z + f_2]} \qquad (2\text{-}16)$$

where

k_g = gain constant = k_{g5}/k_{g6}

$k_{g5} = k_p T + k_v J$

$k_{g6} = B^2$

$b_1 = TB + J(\beta - 1)$

$b_2 = TB\beta + J(\beta - 1)$

$f_1 = \{ k_p b_1 - B^2(1 + \beta) + k_v JB(1 - \beta) \}/k_{g6}$

$f_2 = \{ B^2\beta + k_v JB(1 - \beta) - k_p b_2 \}/k_{g6}$

$g = \{ -k_v J \}/k_{g5}$

III. STATE SPACE CONTROL OF THE ARM

A: State Space Concepts

1. GENERAL EQUATIONS

The state and output equations for a continuous, linear, constant coefficient system are given below.

$$dx(t)/dt = F\,x(t) + G\,u(t)$$

$$y(t) = H\,x(t) + D\,u(t) \qquad (3\text{-}1)$$

where

x(t) = state vector (n x 1)
u(t) = control vector (p x 1)
y(t) = output vector (q x 1)
F = system matrix (n x n)
G = control distribution matrix (n x p)
H = output matrix (q x n)
D = feedthrough matrix (q x p)

If the system is time invariant, the matrices F, G, H, and D become constant matrices. In most cases, the feedthrough matrix D is not needed and it will be omitted in the following discussion.

The system described above can also be represented, at discrete instants of time (t=kT where k=0, 1, 2, ... and T is the sampling period), by a set of difference equations, as shown in equations (3-2).

$$x(k+1) = \Phi\,x(k) + \Gamma\,u(k)$$

$$y(k) = H\,x(k) \qquad (3\text{-}2)$$

where

Φ = discrete system matrix (n x n) $= e^{FT} = \{I + FT + \frac{F^2T^2}{2!} + \ldots\}$

and

Γ = discrete control distribution matrix (n x p)

$$= \int_0^T e^{F\alpha}\, da\, G$$

$$= \{IT + \frac{FT^2}{2!} + \frac{F^2T^3}{3!} + \ldots\}G$$

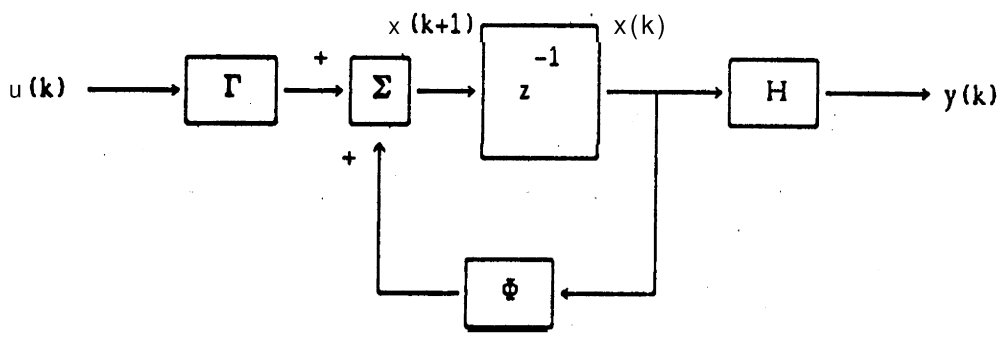
A diagram of the discrett state space system described above is shown in Figure 3-1.

The z-transform transfer function, derived from the difference equations (3-2), is shown below.

$$H(z) = H[zI - \Phi]^{-1}\Gamma \qquad (3\text{-}3)$$

The characteristic equation of the transfer function given above is

$$\det [zI - \Phi] = 0 \qquad (3\text{-}4)$$



where z^{-1} represents a delay of one sampling period

Figure 3-1: Discrete State Space System Representation

2. CONTROLLER DESIGN

A diagram of a discrete state space controller with state feedback is shown in Figure 3-2. The general state feedback equation is given by

$$u(k) = -K\,x(k) + u_o(k) \tag{3-5}$$

where

K = feedback gain matrix (p x n)
u,(k) = system input vector (p x 1)

If the feedback equation (3-5) is substituted into the state difference equations in equations (3-2), the following state difference equation is obtained.

$$x(k+1) = [\Phi - \Gamma K]\,x(k) + \Gamma u_o(k) \tag{3-6}$$

The characteristic equation obtained from equation (3-6) is

$$\det\,[zI - \Phi + \Gamma K] = 0 \tag{3-7}$$

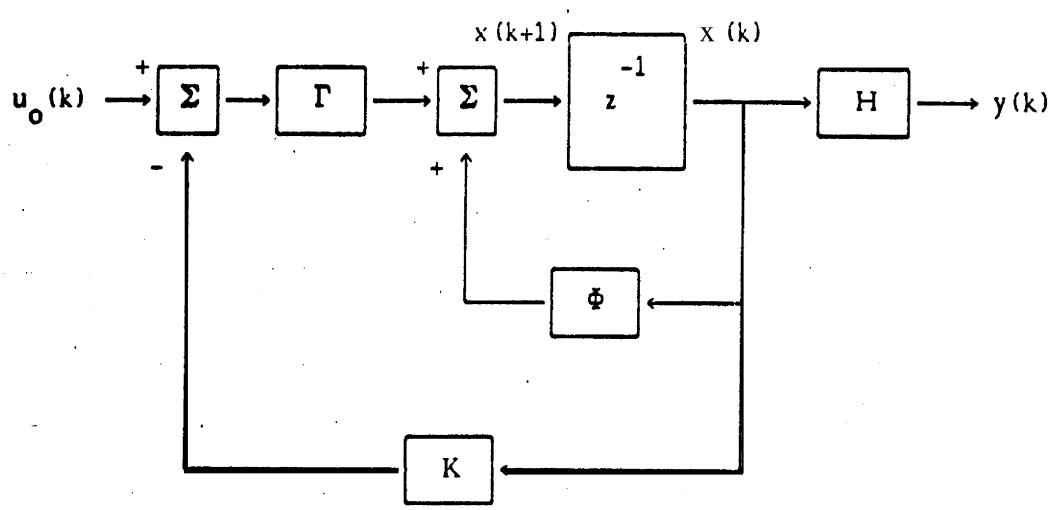


Figure 3-2: Discrete State Space System With State Feedback

A system is defined as "controllable" if its controllability matrix,

$$C = [\ \Gamma\ \ \Gamma\Phi\ \ \Gamma\Phi^2\ \ldots\ \Gamma\Phi^{n-1}\ ]$$

is **nonsingular.** If a system is controllable, the roots of the system can be positioned at any desired location by choosing the appropriate feedback gain matrix K.

B. Design of a Joint Controller

The differential equation describing the transient response of a servomotor system is repeated below from Section **II-B.** The coulomb friction term has been omitted, however, since the friction compensation used in the motor torque equation minimizes the effects of the coulomb friction.

$$T(t) = J\frac{d^2\theta_a(t)}{dt} + B\frac{d\theta_a(t)}{dt} \qquad (3\text{-}8)$$

If position and velocity are chosen as states, the matrices describing a single joint are given by

$$x = \begin{bmatrix} \theta_a(t) \\ \omega_a(t) \end{bmatrix}, \qquad F = \begin{bmatrix} 0 & 1 \\ 0 & -\tau \end{bmatrix}, \qquad G = \begin{bmatrix} 0 \\ \kappa \end{bmatrix}$$

$$u = [\ \kappa T(t)\ ], \qquad H = [\ 1\ \ 0\ ]$$

where

θ_a = actual position (rad)
ω_a = actual velocity (rad/sec)
T(t) = motor torque input (oz-in)
τ = B/J (rad/sec)
κ = 1/ J (1 /(oz-in-sec^2))
B = viscous damping constant (oz-in-set)
J = inertia (oz-in-set^2)
t = continuous time

The discrete difference equations for a single joint are given by

$$x(k+1) = \Phi\, x(k) + \Gamma\, 'I'(k)$$

$$y(k) = H\, x(k) \qquad (3\text{-}9)$$

where

$$x = \begin{bmatrix} \theta_a(k) \\ \omega_a(k) \end{bmatrix}, \quad \Phi = \begin{bmatrix} 1 & \gamma \\ 0 & \beta \end{bmatrix}, \quad \Gamma = \begin{bmatrix} \kappa\mu \\ \kappa\gamma \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

and

$$\beta = e^{-T/\tau}$$
$$\gamma = (1 - \beta)/\tau$$
$$\mu = (T - \gamma)/\tau$$

The motor torque input was given in Section II-A as

$$T(k) = J(k)\alpha_c(k) + G(k) + F - k_c k_p [\theta_a(k) - \theta_c(k)]$$
$$- k_c k_v J(k) [\omega_a(k) - \omega_c(k)] - k_c k_i \sum_{j=1}^{k} [\theta_a(j) - \theta_c(j)]/J(j) \qquad (2\text{-}1)$$

The gravity loading and the friction compensation terms are updated by the computer during motions, so G(k) and F do not need to be included in the joint model. The inertia, J(k), will be assumed to be a constant, although on some of, the joints, it is a time varying parameter which depends on the configuration of the arm. The effect of inertia variations will be examined in Section V-B. Taking the z transform of the remaining terms in equation (2-1) gives

$$T(z) = \frac{J(z-1)^2\theta_c(z)}{(Tz)^2} - k_c\left\{k_p + k_v\frac{J(z-1)}{Tz} + \frac{k_i z}{J(z-1)}\right\}[\theta_a(z) - \theta_c(z)] \qquad (3\text{-}10)$$

A block diagram of the system described above is given in Figure 3-3.

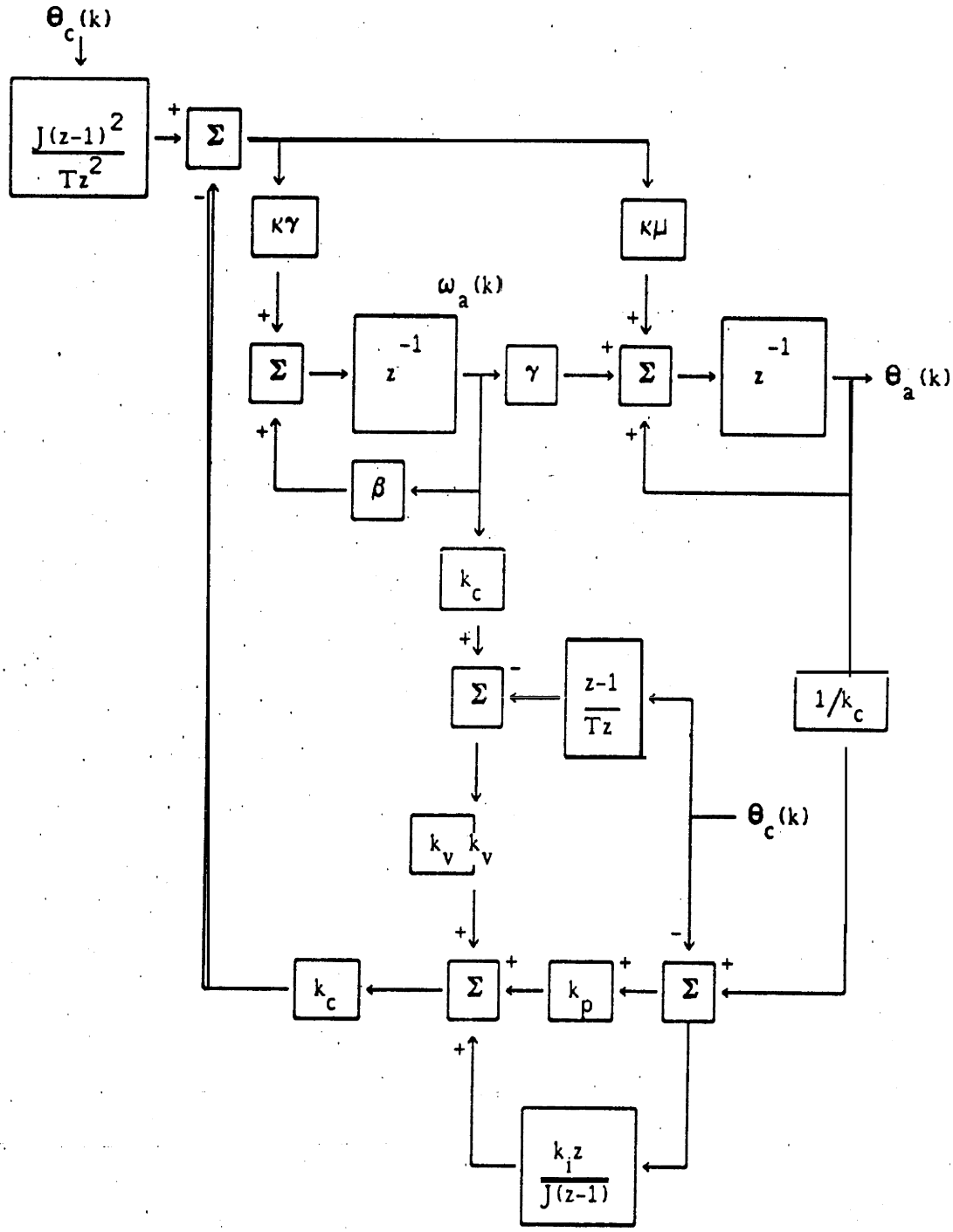


Figure 3-3: Two State Joint Model

By adding three more states, equation (3-10) can be combined with equations (3-9) to give a new set of difference equations:

$$x(k+1) = [\Phi - \Gamma K]\, x(k) + \Gamma \theta_c(k)$$

$$y(k) = H\, x(k) \qquad (3\text{-}11)$$

The new x, Φ, Γ, H, K, and u matrices are given by

$$x = \begin{bmatrix} x_3(k) \\ \theta_a(k) \\ \omega_a(k) \\ x_2(k) \\ x_1(k) \end{bmatrix}, \quad \Phi = \begin{bmatrix} 1 & 1/k_c & 0 & 0 & 0 \\ 0 & 1 & \gamma & 0 & 0 \\ 0 & 0 & \beta & 0 & 0 \\ & & & & \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \Gamma = \begin{bmatrix} 0 & -1 \\ k_c K \mu & 0 \\ k_c K \gamma & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$u = \begin{bmatrix} (h_0 + k_i)\theta_c \\ \theta_c \end{bmatrix}, \quad K = \begin{bmatrix} k_i K & (k_i K + k_p)/k & (k_v J)/k_c & h_2 & h \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

and

$$h_0 = (k_p T^2 + k_v T J + J)/T^2$$
$$h_1 = -(k_v T J + 2J)/T^2$$
$$h_2 = J/T^2$$

A diagram showing the additional states is given in Figure 3-4.

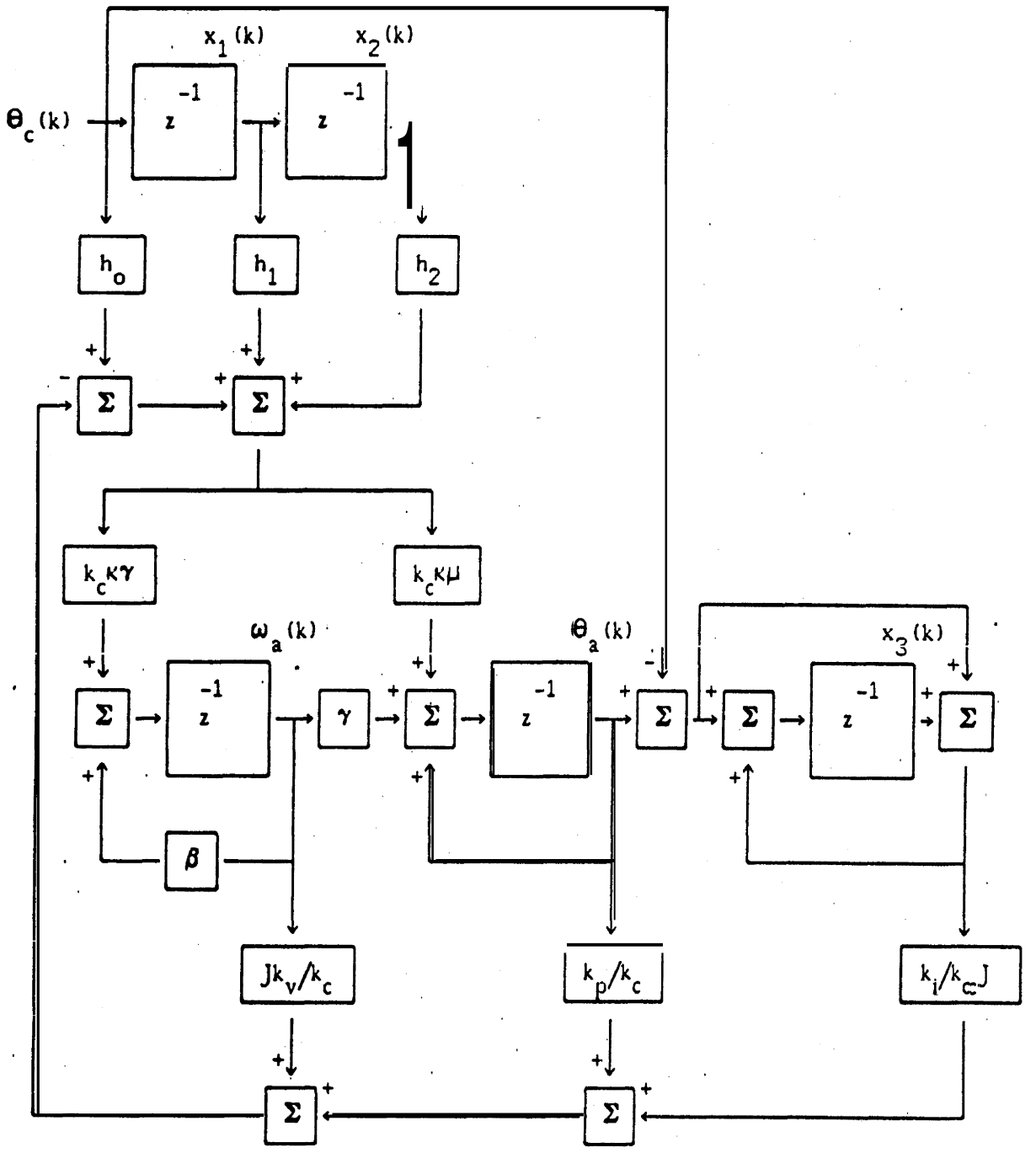


Figure 3-4: Five State Joint Model

The transfer function for the closed, loop system given above can be calculated using equation (3-6) which is repeated below.

$$H(z) = \frac{\theta_a(z)}{\theta_c(z)} = H\,[\,zI - \Phi + \Gamma K\,]^{-1}\,\Gamma \qquad (3\text{-}6)$$

Expanding equation (3-6) gives

$$H(z) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} z-1 & -1/k_c & 0 & 0 & 0 \\ k_c k_i \mu\kappa^2 & z-1+k_p\mu\kappa+k_i\mu\kappa^2 & k_v\mu-\gamma & k_c h_2 \mu\kappa & k_c h_1 \mu\kappa \\ k_c k_i \gamma\kappa^2 & k_p\gamma\kappa+k_i\mu\kappa^2 & z-\beta+k_v\gamma & k_c h_2 \gamma\kappa & k_c h_1 \gamma\kappa \\ 0 & 0 & 0 & z & -1 \\ 0 & 0 & 0 & 0 & z \end{bmatrix}^{-1} \times \begin{bmatrix} 0 & -1 \\ k_c\kappa\mu & 0 \\ k_c\kappa\gamma & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} h_0+k_i\kappa \\ 1 \end{bmatrix} \qquad (3\text{-}12)$$

Recking equation (3-12) gives

$$H(z) = \frac{\theta_a(z)}{\theta_c(z)} = \frac{NUM}{DEN}$$

where

$$NUM = k_c\kappa\,[\,k_i\kappa z^3 + h_0(z-1)z^2 + h_1(z-1)z - h_2(z-1)\,]\,[\,(z-\beta+k_v\gamma)\mu + (\gamma-k_v\mu)\gamma\,]$$

$$DEN = z^2\{[\,z-\beta+k_v\gamma\,][\,k_i\mu\kappa^2 + (z-1)(z-1+k_p\mu\kappa+k_i\mu\kappa^2)\,] + \kappa[\gamma-k_v\mu\,][\,k_i\gamma\kappa + (z-1)(k_p\gamma+k_i\gamma\kappa)\,]\}$$

Further reduction gives

$$H(z) = \frac{k_c k_g [b_1 z - b_2][z^3 + d_1 z^2 + d_2 z + d_3]}{(Tz)^2 [z^3 + c_1 z^2 + c_2 z + c_3]} \quad (3\text{-}13)$$

where

k_g = gain constant = k_{g1}/k_{g2}

$k_{g1} = T^2[k_i + k_p J] + J^2[k_v T + 1]$

$k_{g2} = JB^2$

$b_1 = TB + J(\beta - 1)$

$b_2 = TB\beta + J(\beta - 1)$

$c_1 = \{b_1[k_i + k_p J] - JB[B(2 + \beta) + k_v J(\beta - 1)]\}/k_{g2}$

$c_2 = \{JB[B + 2B\beta + 2kv\ J(\beta - 1)] - b_2[ki + k_p J] - k_p J b_1\}/k_{g2}$

$c_3 = \{k_p J b_2 - JB[B\beta + k_v J(\beta - 1)]\}/k_{g2}$

$d_1 = \{-J[k_p T^2 + J[2k_v T + 3]]\}/k_{g1}$

$d_2 = \{J^2[k_v T + 3]\}/k_{g1}$

$d_3 = \{-J^2\}/k_{g1}$

This is **the same** transfer that was obtained in Section II-B using **a** classical analysis. **Classical** and state space methods of analysis often give the **same** transfer function, especially for single input - single output systems. The transfer function above has two closed loop poles at the origin of the z-plane, one real pole, and two poles which can be either real or complex. If 10 feedback gains had been used in the feedback gain vector K, instead of five, it 'would **have** been. possible to position the five poles at any desired location. Only five gains were used, 'however, since there is **no** advantage in altering the positions of the two poles at the origin. ,

The feedback gains required to obtain the desired joint characteristic equation given by

$$z^2[z^3 + \alpha_1 z^2 + \alpha_2 z + \alpha_3] = 0 \quad (3\text{-}14)$$

can be computed using equation (3-15).

$$K = B^{-1}[A - C] \quad (3\text{-}15)$$

where

$$A = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \qquad B = \frac{1}{JB^2} \begin{bmatrix} -J^2B(\beta-1) & b_1J & b_1 \\ 2J^2B(\beta-1) & -(b_1+b_2)J & -b_2 \\ -J^2B(\beta-1) & b_2J & 0 \end{bmatrix}$$

$$K = \begin{bmatrix} k_v \\ k_v \\ k_i \end{bmatrix} \qquad C = \begin{bmatrix} -(2+\beta) \\ 1+2\beta \\ -\beta \end{bmatrix}$$

The computation of feedback gains will be discussed in further detail in Chapter VI.

IV. DETERMINATION OF ARM PARAMETERS

A. Summary of Measurement *Techniques

The differential equation describing the transient response of a single joint of the arm is repeated below from Section II-B.

$$T(t) = J \frac{d^2\theta_a(t)}{dt} + B \frac{d\theta_a(t)}{dt} + F \qquad (2\text{-}4)$$

where

T(t) = motor torque input (oz-in)
$\theta_a(t)$ = actual joint 'position (rad)

J = inertia of motor, arm, and load (or-in-set^2)
B = viscous damping of motor and arm (ot-in-set)
F = coulomb friction of motor and arm (oz-in)

It can be seen from equation (2-4) that inertia, damping, and friction are the three arm parameters which have the most predominant effect on the arm's time response. Inertia can be calculated from a knowledge of the mass and relative position of each of the component parts of the arm. An excellent reference on this subject is [BE].

The arm's coulomb friction and viscous damping are difficult to measure because they are position dependent. The position dependency is illustrated in Figure 4-1 where the velocity of each of the six joints has been plotted as a function of position for a constant torque step input. If friction and damping were independent of position, the velocity would be constant after an intial period of acceleration. In Figure 4-1, it can be seen that the velocities of the joints are not constant, even after the acceleration period has ended. When the motions plotted in Figure 4-1 were repeated, the velocities were found 'to be repeatable functions of position.

The position dependency complicates the measurement process, but using the average values of the friction and damping in the motor torque equation and the arm model gives an accurate prediction of the arm's transient response. The insensitivity of the model to friction and damping is a result of the fact that the velocity feedback overides the effects of errors in the viscous damping, and the position and integral feedback minimize the effect of errors in the coulomb friction.

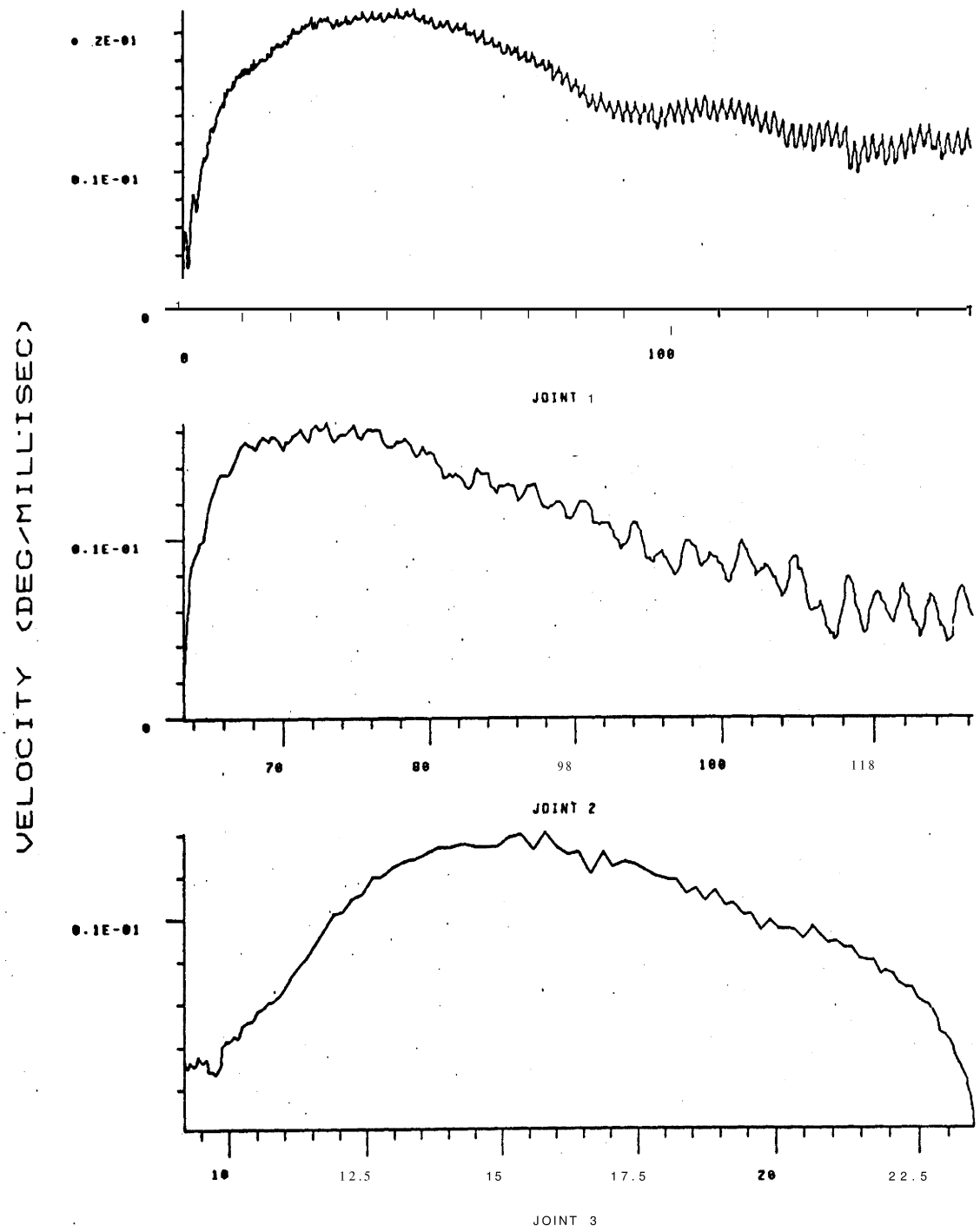


Figure **4-1: Position Dependence** of **Friction**

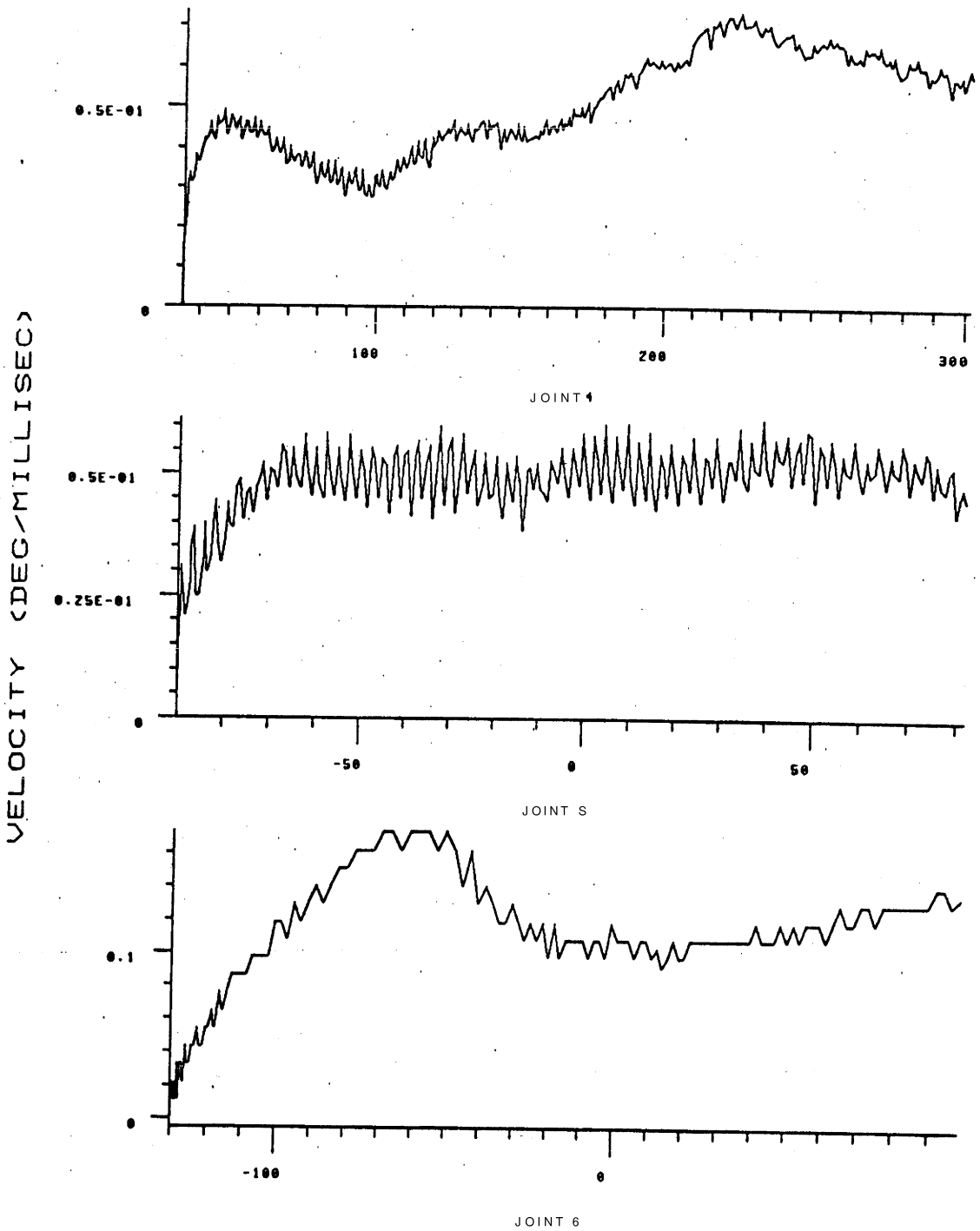


Figure 4-1 (continued): Position Dependence of Friction

A measurement of the arm's average coulomb friction can be obtained using a method developed by Richard Paul and Bruce Shimano at the Artificial Intelligence Laboratory. The method consists of applying a constant motor torque to a joint, and then measuring:

- the restraining force required to maintain a constant velocity in the direction of applied torque
- the force required to pull the arm at a constant velocity in a direction opposite to the applied motor torque.

The friction is determined by dividing the difference between the two force measurements by two.

'Five methods, which have been investigated for measuring joint viscous damping, are listed below.

1. Terminal Velocity - Torque Step Input
2. Least Squares Fit - Torque Step Input
3. Transient Response - Position Step Input
4. Least Squares Fit - Position Step Input
5. Bode Plot Analysis - Position Sine Wave Input

Some of these methods also measure inertia or coulomb friction as well. Of the methods listed above, methods #3 and #5 give the most consistent results and are the easiest to use. All of the methods are summarized briefly below, but tests #3 and #5 are explained in detail in the following sections.

1. TERMINAL VELOCITY - TORQUE STEP INPUT

The feedback and feedforward terms in the motor torque equation are deleted and the joint is given a step input in torque. After the joint's initial acceleration, the velocity of the joint is given by equation (4-1).

$$\omega_a(t) = (T - F)/k_c B \qquad (4\text{-}1)$$

where k_c = constant to convert degrees to radians = .01745

Both the viscous damping and the coulomb friction can be determined as functions of position by using different values of torque on successive runs. This test does not work well on joints which have a limited amount of rotation. The applied torque must be kept very' low to allow the joints to reach their terminal velocities, yet at low torques, the erratic effects of coulomb friction mask the real terminal velocities.

2. LEAST SQUARES FIT - TORQUE STEP INPUT

The feedforward and feedback terms are deleted from the motor torque equation and the joint is given a step input in torque. A least squares curve fit is then made between equations (4-2) and (4-3) and the joint's actual transient response.

$$\theta_a(t) = K\,[t - \tau\,(1 - e^{-t/\tau})] \tag{4-2}$$

$$\omega_a(t) = K\,(1 - e^{-t/\tau}) \tag{4-3}$$

where

$$K = (T - F)/B$$
$$\tau = J/B$$

This test also does not work well for joints which have a limited amount of rotation. The curvature of the velocity versus time response of these joints is small and the values obtained from the least squares fit vary with the applied motor torque.

3. TRANSIENT RESPONSE - POSITION STEP INPUT

The feedforward and the velocity and integral feedback terms are deleted from the motor torque equation. The joint is then, given a step input in position and its proportional gain is varied until the joint's response to the step is underdamped. The joint's viscous damping and inertia can be computed from the peak time at which the arm begins to reverse its motion and the magnitude of the peak overshoot. This test is the easiest to use of the five, and it produces repeatable and accurate results. It is discussed in detail in Section IV-B.

4. LEAST SQUARES FIT - POSITION STEP INPUT

The feedforward and the velocity and integral feedback terms are deleted from the motor torque equation. The joint is then given a step input in position and and its proportional gain is varied until the joint's response to the step is underdamped. A least squares fit is made between the output of equation (4-3) and the joint's actual transient response. (Equation (4-4) is derived in Section IV-B-I.)

$$\theta_a(t) = k_c\left[1 - e^{-\zeta\omega_n t}\left(\cos\omega_d t + \left(\frac{\zeta}{(1-\zeta^2)^{1/2}}\right)\sin\omega_d t\right)\right] \tag{4-4}$$

This test is the best method to use when extremely accurate values are needed for inertia and damping. If the coulomb friction compensation is adjusted until the transient response is

exactly second order, this test will give the most accurate results of the five tests presented in this section. The added difficulties incurred in determining the exact coulomb friction compensation, however, make this test difficult to use.

5. BODE PLOT ANALYSIS - POSITION SINE WAVE INPUT

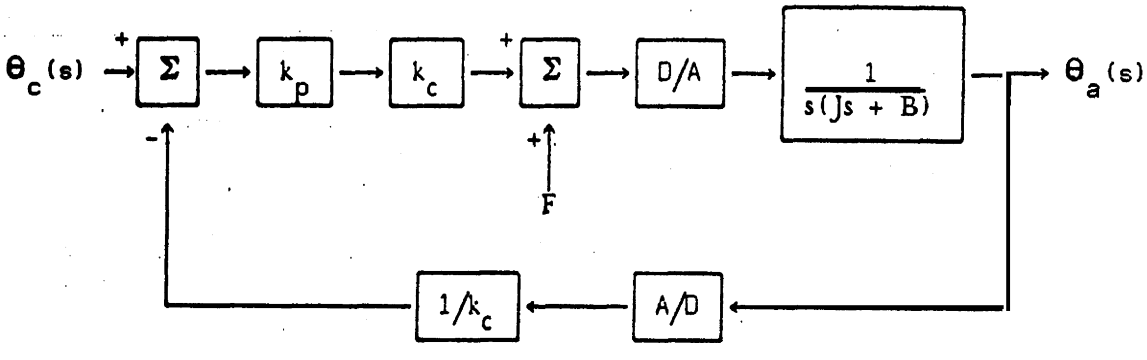
The feedforward and the velocity and integral feedback terms are deleted from the motor torque equation and the joint is excited with a sine wave position command. The frequency which produces the largest peak to peak displacement is the joint's resonant frequency and can be used to compute the joint inertia. The viscous damping can be calculated from the magnitude of the displacement at the resonant frequency.

This test produces accurate and repeatable results and is moderately easy to use. It is a good backup test for verifying the results obtained from the Test #3.

B. Detailed Description of Preferred Measurement Techniques

I, TRANSIENT RESPONSE - POSITION STEP INPUT

This method uses the peak time and peak dvershoot of a joint's transient response to determine the joint's inertia and viscous damping. The block diagram of the system configuration used for this test is given in Figure 4-2.



where

k_p = proportional feedback constant

k_c = constant to convert degrees to radians = .01745

Figure 4-2: Configuration for Transient Response Test

The joint is given a step input in position and its proportional gain is varied until the joint's step response is underdamped. The peak time at which the arm begins to reverse its motion, and the magnitude of the peak overshoot are then noted for step inputs of various sizes. A typical motion is shown in Figure 4-3.

'Since the present 60 hz sampling rate is more than 15 times higher than the bandwidth of the joint configuration shown in Figure 4-2, a continuous transfer function can be used to model the arm's closed loop transient response. The transfer function can be determined by inspection of Figure 4-2 and is given by

$$H(s) = \frac{\theta_a(s)}{\theta_c(s)} = \frac{k_c k_p}{Js^2 + Bs + k_p} \qquad (4\text{-}5)$$

For a step input of magnitude 1.0, the response of the arm is given by

$$\theta_a(s) = \frac{k\,k_p}{s(Js^2 + Bs + k_p)}$$

$$= \frac{k_c \omega_n^2}{s(s^2 + 2\zeta\omega_n s + \omega_n^2)} \qquad (4\text{-}6)$$

where

$$\omega_n^2 = k_p/J \qquad (4\text{-}7)$$

$$2\zeta\omega_n = B/J \qquad (4\text{-}8)$$

ζ = damping ratio

ω_n = undamped natural frequency (rad/sec)

The time domain step response, $\theta_a(t)$, can be calculated by taking the inverse Laplace transform of equation (4-6).

$$\theta_a(t) = k_c[1 - e^{-\zeta\omega_n t}(\cos\omega_d t + \{\frac{\zeta}{(1-\zeta^2)^{1/2}}\}\sin\omega_d t)] \qquad (4\text{-}9)$$

where $\omega_d = \omega_n [1 - \zeta^2 3^{1/2}$.

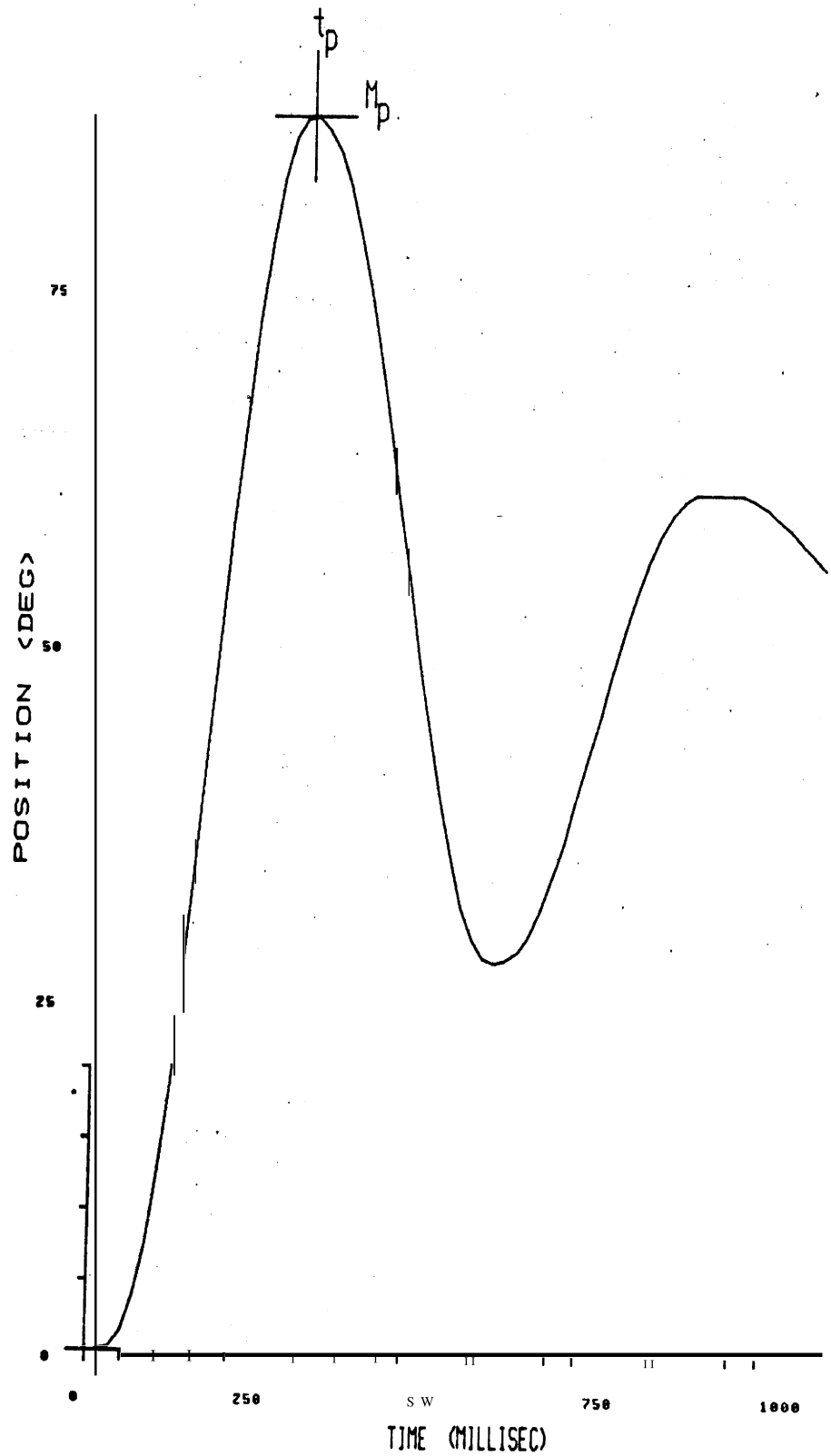


Figure 4-3: Typical Step Response . Transient Response Test

The peak time t_p can be determined by taking the derivative of $\theta_a(t)$ given in equation (4-9) and equating it to zero. This gives

$$t_p = \pi/\omega_d \qquad (4\text{-}10)$$

The peak overshoot, M_p, can be calculated by evaluating $\theta_a(t)$ at t_p.

$$M_p = \exp[-\pi\zeta/(1-\zeta^2)^{1/2}] \qquad (4\text{-}11)$$

Since t_p and M_p can be measured experimentally, equations (4-10) and (4-1 1) can be rearranged to give ζ and ω_n as functions of t_p and M_p. The inertia and damping can. then be determined from ζ and ω_n by rearranging equations (4-7) and (4-8).

$$\zeta = \left[\frac{(\ln(1/M_p))^2}{\pi^2 + (\ln(1/M_p))^2}\right]^{1/2} \qquad (4\text{-}12)$$

$$\omega_n = \frac{\pi}{t_p(1-\zeta^2)^{1/2}} \qquad (4\text{-}13)$$

$$J = k_p/\omega_n^2 \qquad (4\text{-}14)$$

$$B = 2\zeta\omega_n J \qquad (4\text{-}15)$$

A more detailed derivation of equations (4-6) through (4-1 1) can be found in reference [OG]. The results obtained using this test are shown in Table 4-l.

2. Bode Plot Analysis - Position Sine Wave Input

For this method, the joint is excited with a sine wave position command and the peak to peak displacement of the joint's response is recorded as a function of the frequency of the sine wave. The system configuration for this test is identical to that given for the closed loop transient response test in Figure 4-2. The prbportional gain should be adjusted to make the joint's transient response underdamped. (ζ should be approximately 0.1)

Joint	Step Input (deg)	k_p (sec)	t_p	θ_p (deg)	B (oz-in-sec)	J (oz-in-sec^2)
1	20	10K	0.616	41.5	87	384
1	30	10K	0.616	62.4	94	384
1	40	10K	0.616	84.3	127	384
1	F = 1150 oz-in			AVG:	102	384
2	30	10K	0.712	4s. 1	709	501
2	40	10K	0.696	65.9	610	482
2	30	20K	0.512	52.5	688	525
2	F = 1000 oz-in			AVC:	669	503
3	5	7	6.880	S.8	0.34	0.55
3	6	7	0.896	11.1,	0.2 1	0.57
3	7	7	0.864	13.1	0.1s	0.53
3	F = 12 oz-in			AVC:	0.24	0.55
4	40	1K	0.304	67.5	22.8	9.2
4	50	1K	0.304	86.9	18.6	9.3
4	60	1K	0.304	107.4	14.5	9.3
4	F = 110 oz-in			AVG:	18.6	9.3
5	40	400	0.67'2	59.6	37.0	I 7.4
5	50	400	0.664	79.5	27.6	17.4
5	55	4.00	0.692	90.9	23.4	19.1
5	F = 30 or-in			AVG:	29.3	18.1
6	50	300	0.440	75. I	17.6	5.6
6	60	300	0.4 32	95.2	13.7	5.6
6	70	300	0.4 32	120.3	8.6	5.6
6	F = 23 oz-in			AVG:	13.3	5.6

where F = compensation for joint coulomb friction.

Table 4-1: Results - Transient Response Test

The computations for this test can be performed in the s-plane since the sampling rate is 15 times the maximum joint bandwidth. The transfer function for this configuration is repeated below from Section IV-B-1.

$$H(s) = \frac{\theta_a(s)}{\theta_c(s)} = \frac{k_c k_p}{Js^2 + Bs + k_p} \tag{4-5}$$

$$= \frac{k_c \omega_n^2}{(s^2 + 2\zeta\omega_n s + \omega_n^2)} \tag{4-16}$$

where

$$\omega_n^2 = k_p/J \tag{4-7}$$

$$2\zeta\omega_n = B/J \tag{4-8}$$

Then

$$H(j\omega) = \frac{k}{1 + 2\zeta(j\omega/\omega_n) + (j\omega/\omega_n)^2} \tag{4-17}$$

The magnitude of the response is given by

$$|H(j\omega)| = \frac{k}{\{[1 - \omega^2/\omega_n^2]^2 + [2\zeta\omega/\omega_n]^2\}^{1/2}} \tag{4-18}$$

The value of $|H(j\omega)|$ peaks at the resonant frequency of the joint. The resonant frequency can be computed by, taking the derivative of equation (4-18) and equating it to zero. The resonant frequency is then given by

$$\omega_r = \omega_n [1 - 2\zeta^2]^{1/2} \tag{4-19}$$

Since the proportional gain for this test was selected so that the system's response would be highly underdamped, ζ is very low and

$$\omega_r \approx \omega_n \tag{4-20}$$

The inertia of a joint can be determined from the joint's transient response using equation (4-7).

$$J = k_p/\omega_n^2 \approx k_p/\omega_r^2 \qquad (4\text{-}21)$$

The magnitude of the response at $\omega = \omega_r$ can be found by substituting equation (4-19) into equation (4-18).

$$M_r = 1/[\,2\zeta\,(\,1-\zeta^2\,)^{1/2}\,] \qquad (4\text{-}22)$$

Thus

$$\zeta = [\,.5 - (\,1 - 1/M_r^2\,)^{1/2}\,]^{1/2} \qquad (4\text{-}23)$$

The viscous damping can then be determined from equation (4-8).

$$B = 2\zeta J\omega_r \qquad (4\text{-}24)$$

A detailed derivation of equations (4-16) through (4-19) can be obtained from reference [OG]. The experimentally determined values of inertia and viscous damping have been tabulated in Table 4-2.

Joint	Sine Input (deg P-P)	k_p (sec)	t_p	θ_p (deg P-P)	B (oz-in-set)	J (oz-in-set^2)
1	6	7.5K	S.425	15.9	272	386
2	6	50K	0.640	15.1	988	519
3	1	10	1.350	21.9	0.28	0.46
4	10	2K	0.4 00	15.8	24.2	8.1
5	10	1K	0.800	16.9	23.7	16.2
6	8	1K	0.500	18.8	15.2	6.3

Table 4-2: Results . Bode Plot Analysis

3. *COMPARISON OF TEST RESULTS*

The results of the Transient Response Test and the Bode Plot Analysis are listed in Table 4-3. For purposes of comparison, the table also shows inertias computed by the present run-time program using equations given in reference [BE] with parameter values for the Stanford. Arm.

The results of these tests will be used in the following chapters for computing the positions of the closed loop roots of the joints. The tests have also pointed out a discrepancy between the computed inertia and the actual inertia of joint 6 and possibly joint 1. The feedforward compensation on joint 6 will be more effective when the constants used to compute its inertia are updated.

Joint	METHOD	B (oz-in-set)	J (oz-in-set^2)
1	TR	102	384
1	BP	272	386
1	BE	-	228
2	TR	669	503
2	BP	988	519
2	BE	-	457
3	TR	0.24	0.55
3	BP	0.28	0.46
3	BE	-	0.48
4	TR	18.6	9.3
4	BP	24.2	8.1
4	BE	-	10.4
5	TR	29.3	18.1
5	BP	23.7	16.2
5	BE	-	12.1
6	TR	13.3	5.6
6	BP	15.2	6.3
6	BE	-	0.89

where

TR = Transient Response Test
BP = Bode Plot Analysis
BE = 'Run-time inertias calculated using equations from reference [BE]

Table 4-3: Comparison of Test Results

V. STABILITY ANALYSIS

A. Root Locus Analysis

A root locus analysis can be used to indicate the stability of the arm as a function of its feedback gains, inertia, *or* sampling rate. In the analysis that follows, the arm has been analyzed in the z-plane where the stability of the arm decreases as its closed loop poles move away from the origin of the z-plane toward the unit circle. When the roots move outside of the unit circle, the arm becomes unstable.

The closed loop transfer function for the arm control system given in equation (2-14) is repeated below.

$$H(z) = \frac{\theta_a(z)}{\theta_c(z)} = \frac{k_c k_g [b_1 z - b_2][z^3 + d\ z^2 + d\ z + d_3]}{(Tz)^{23}[z + c_1^2 z + c_2 z + c_3 J} \qquad (2\text{-}14)$$

where

k_g = gain constant $= k_{g1}/k_{g2}$

$k_{g1} = T^2[k_i + k_p J] + J^2[k_v T + 1]$

$k_{g2} = JB^2$

$b_1 = TB + J(\beta - 1)$

$b_2 = TB\beta + J(\beta - 1)$

$c_1 = (b_1[ki + k_p J] - JB[B(2 + \beta) + k_v J(\beta - 1)]\}/k_{g2}$

$c_2 = (JB[B + 2B\beta + 2k_v J(\beta - 1)] - b_2[k_i + k_p J] - k_p J b_1\}/k_{g2}$

$c_3 = \{k_p J b_2 - JB[B\beta + k_v J(\beta - 1)]\}/k_{g2}$

$d_1 = \{-J[k_p T^2 + J[2k_v T + 3]]\}/k_{g1}$

$d_2 = \{J^2[k_v T + 3\ 3\}/k_{g1}$

$d_3 = \{-J^2\}/k_{g1}$

The parameter values and the present feedback gains for each of the joints are listed in Table 5-1. The values given for viscous damping and inertia are the averages of the experimental measurements listed 'in Table 4-3.

Using these parameter values, the positions of the closed loop poles of each of the joints have been determined. The positions of the closed loop poles for joint 1 are plotted in Figure 5-1A. The far right side of Figure 5-1A has been expanded in Figure 5-1B. In the interest of brevity, this report presents graphical results only for joint 1. Readers interested in seeing the graphical results for the other joints are referred to my thesis.[RO]

Joint	J (oz-in-set^2)	B (oz-in-set)	k_p (oz-in)	k_v (1/sec)	ki (oz^2-in^2-sec^2)
1	385	187	1.5 E5	50	6.0 E5
2	511	829	3.0 E5	70	1.0 E6
3	0.51	0.26	200	30	4.0
4	8.7	21.4	7000	30	3.0 E3
5	17.2	26.5	2000	40	1.0 E3
6	6.0	14.3	rood	100	50.0

where for joint 3 the units are J = oz-set^2/in and B = oz-sec/in.

Table 5-l: Arm Parameter Values

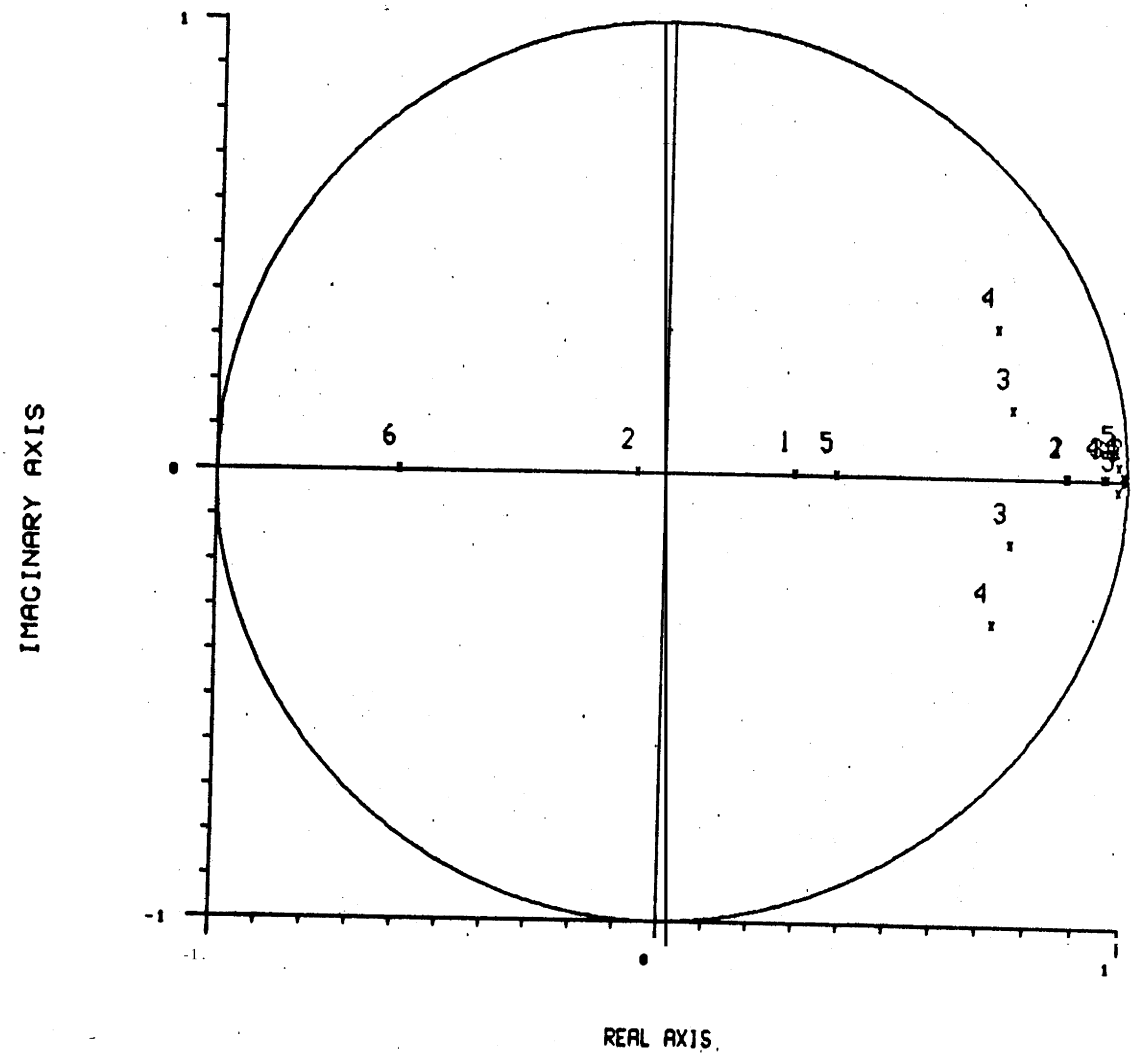
B. Inertia Effects

, The dynamic **response of a joint is** affected by its inertia and the inertia of any load picked up by the arm. To illustrate the **effects** of inertia on the arm's closed **loop poles,** the joint 1 pole **locations have been** plotted in Figure 5-2 for inertias **of** 1, 2, 4, 8, and 16 times the **nominal** joint **inertia** listed in Table 5-1. In this analysis the feedback gains were held fixed at **the values shown in** Table **5-1,** and it was assumed that the inertia terms **in** the motor torque **equation** were updated to include the **additional** inertia. Similar graphs are presented in my thesis (reference **[RO]**) for the remaining 5 joints. These graphs show that the z-plane pole locations 'are **shifted** by inertia variations, **but** all joints remain stable for inertia variations of **at** least a factor of 16 times the nominal inertia.

The inertia of, **joints** 1 and **2** can vary by a factor of two, depending upon the configuration of the **arm. The** effect of this variation on the closed loop poles of joint 1 is roughly equal to the distance between points *1 and *2 on Figure 5-2.

In reference **[BE],** Bejczy shows that the act of picking up **a** 4 lb cube approximately doubles the inertia of **each** of the joints. Thus, the closed loop poles of the joint, when the arm is holding the cube, are shown as the *2 points in Figure 5-2.

In Section **VI-A, a** modification to the present motor torque equation will be discussed that will significantly reduce the effects of inertia demonstrated above.



'Figure **5-1A: Joint** Root Loci

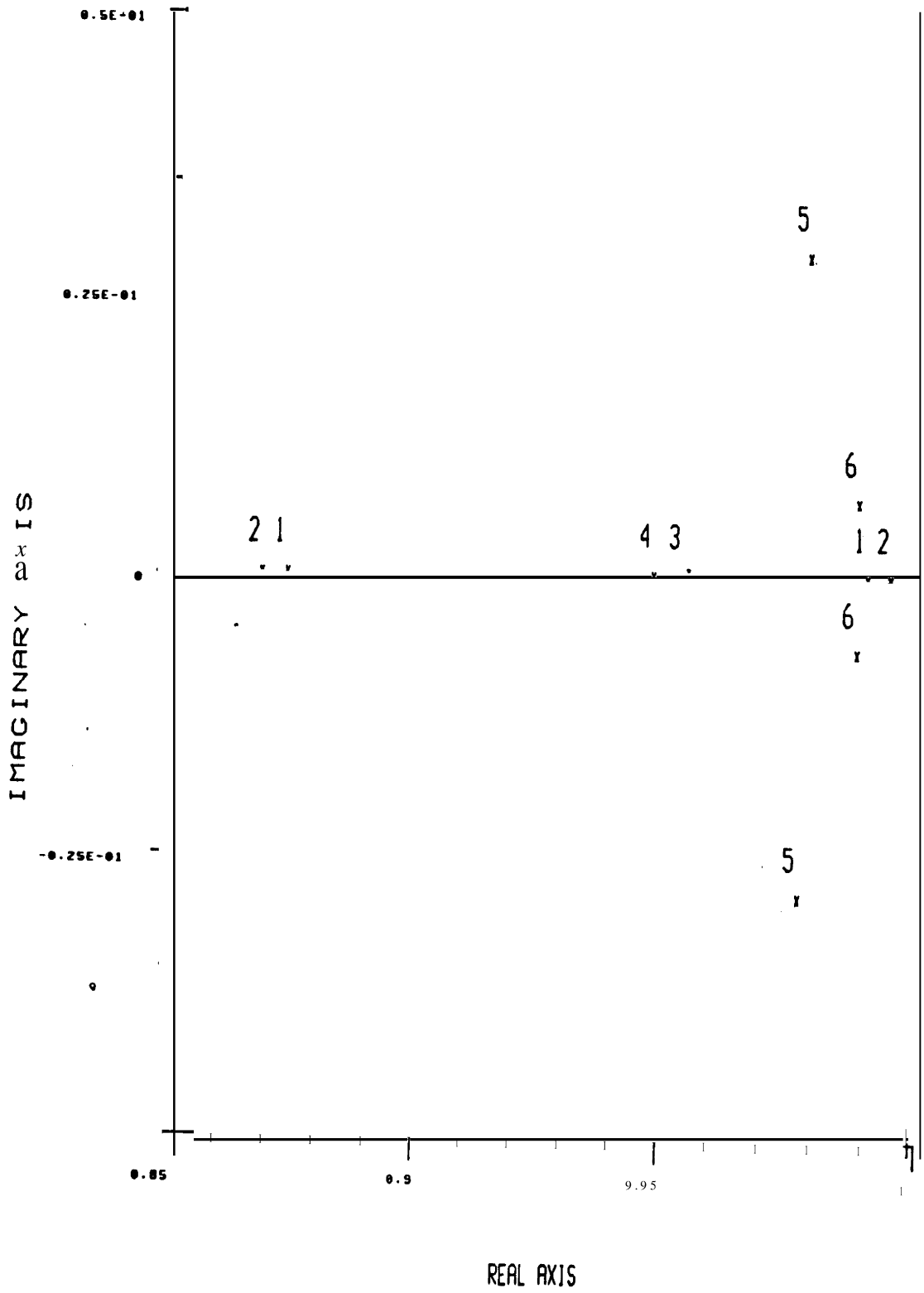


Figure **5-1B: Joint** Root Loci . Expanded Scale

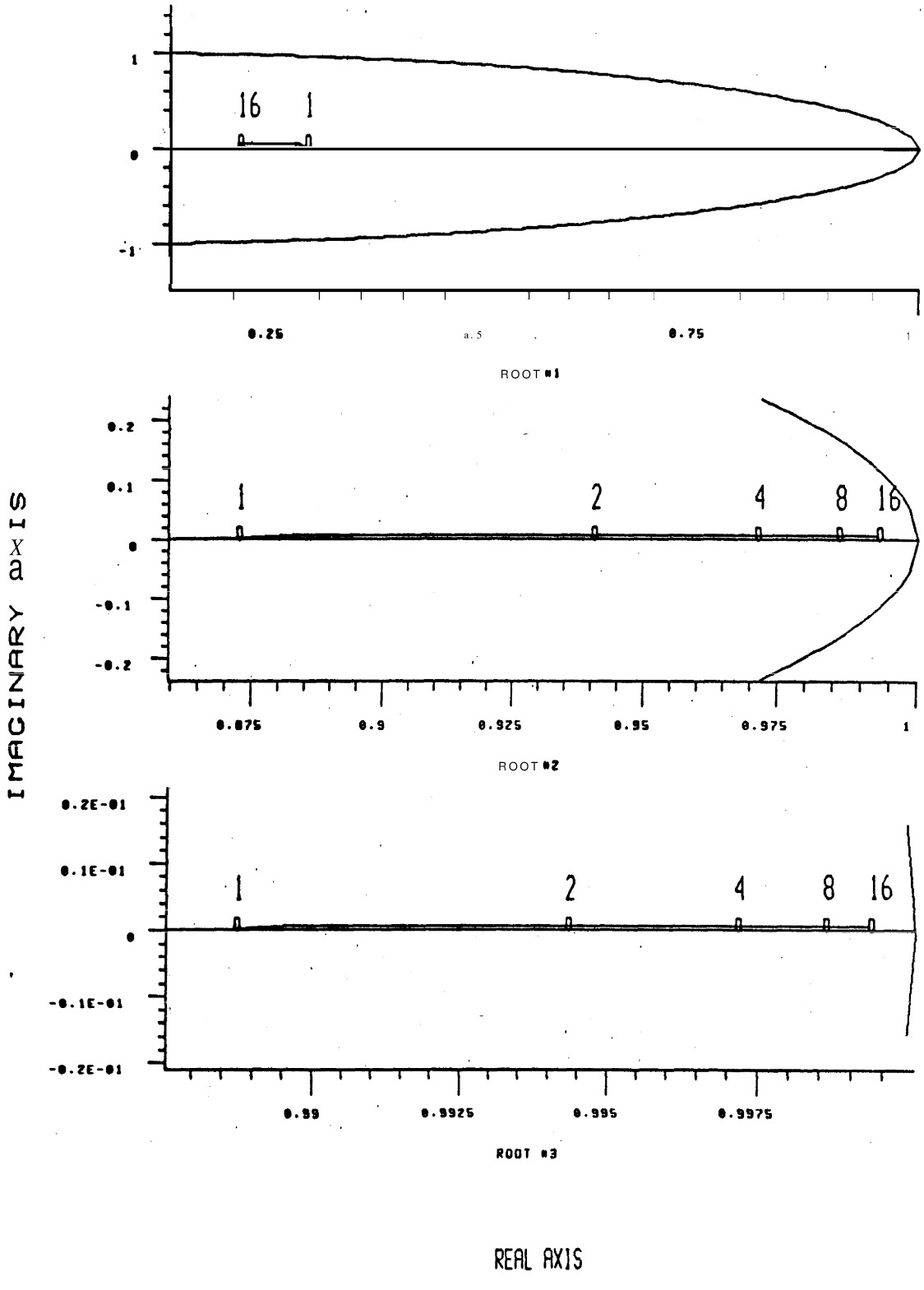


Figure 5-2: Inertia Effects - Joint #1 Root Loci

C. Sampling Rate Sensitivity

The dynamic response of the arm control system is a strong function of the system's sampling rate. Figure 5-3 shows the movement of the closed loop poles of joint 1 as the sampling rate is reduced from 100 hz to 10 hz. All feedback, gains were held fixed at the values specified in Table 5-I. Similar root locus plots for the other joints indicated that all of the joints become unstable before the sampling rate reaches 10 hz.

When the sampling rate sensitivity of the joints was verified by running the joints at reduced, sampling rates, it was found that that the actual joint responses agreed with the root locus plots, except for joint 6. The root locus plots indicated joint 6 would become unstable at 50 hz, yet actual tests on the joint showed that it did not become unstable until the sampling rate fell to between 5 and 10 ht. The anomaly was traced to the error in the computed inertia noted earlier in Table 4-3. The derivative feedback gain in the motor torque equation is multiplied by inertia to reduce the sensitivity of the control system to inertia. The inertia computed for joint 6 was shown in Table 4-3 to be 0.89 oz-in-set^2 while actual measurements showed the inertia to be 6.0 oz-in-set^2. Thus, the actual gain of the derivative feedback term was 6.5 times smaller than expected. The root loci for joint 6 were recomputed using the reduced value of derivative feedback. The new plot showed that the joint will go unstable at 7.5 hz, in good agreement with the experimental findings.

On future arm systems, it would be interesting to investigate the feasibility of changing the sampling rate dynamically, so that the sampling rate would be higher during periods of acceleration and deceleration. It would then be desirable to make the control system as insensitive as possible to variations in the sampling rate. A suggested modification for reducing the present sensitivity to sampling rate variations will be presented in Section VI-B.

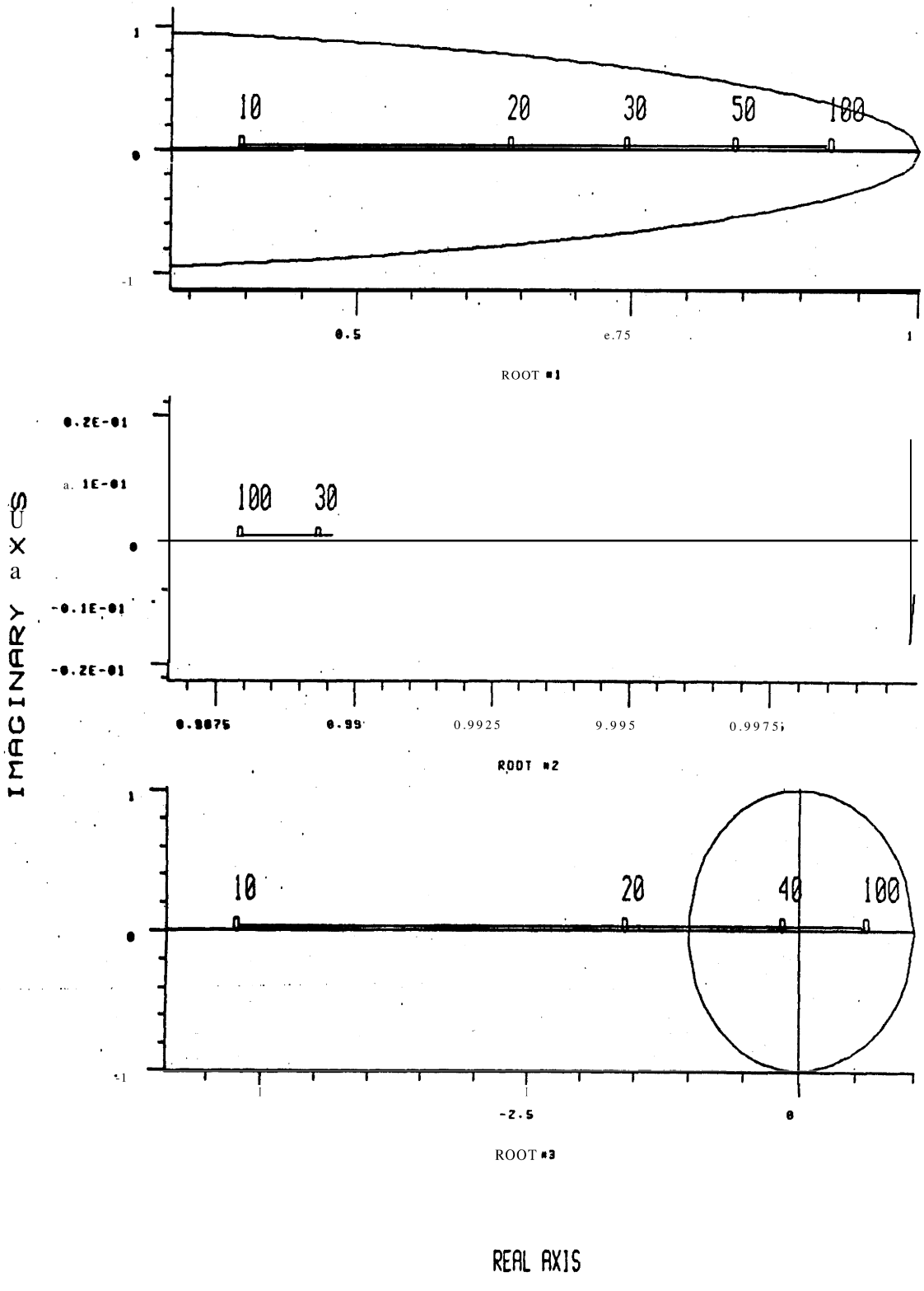


Figure 5-3: Sampling Rate Sensitivity - Joint #1 Root Loci

'VI. RECOMMENDATIONS

A. Reduction of Inertia Effects

In Section V-B; it was shown that the locations of the arm's closed loop poles are affected by changes in inertia. The effect of joint inertia variations can be significantly reduced by rearranging the inertia terms in the joint motor torque equation so that each of the feedback gains is multiplied by the inertia. The feedback terms in the present motor torque equation are given by

$$T(k)_{fb} = -k_c \{ k_p [\theta_a(k) - \theta_c(k)] + k_v J(k) [\omega_a(k) - \omega_c(k)] + k_i \sum_{j=1}^{k} [\theta_a(j) - \theta_c(j)]/J(j) \} \quad (6\text{-}1)$$

If the inertia terms are rearranged to give

$$T(k)_{fb} = -k_c J(k) \{ k_p [\theta_a(k) - \theta_c(k)] + k_v [\omega_a(k) - \omega_c(k)] + k_i \sum_{j-1}^{k} [\theta_a(j) - \theta_c(j)] \} \quad (6\text{-}2)$$

then the joint transfer function becomes

$$H(z) = \frac{\theta_a(z)}{\theta_c(z)} = \frac{kck\ [b_1 z - b_2][z^3 + d_1 z^2 + d_2 z + d_3]}{(Tz)^2 [z^3 + c_1 z^2 + c_2 z + c_3]} \quad (6\text{-}3)$$

where

k_g = gain constant = k_{g1}/k_{g2}

$k_{g1} = T^2 [k_p + k_i] + k_v T + 1$

$k_{g2} = B^2$

$b_1 = TB + J(\beta - 1)$

$b_2 = TB\beta + J(\beta - 1)$

$c_1 = \{ b_1 J [k_p + k_i] - B [B(2 + \beta) + k_v J(\beta - 1)] \}/k_{g2}$

$c_2 = \{ B [B + 2B\beta + 2k_v J(\beta - 1)] - b_2 J [k_p + k_i] - k_p J b_1 \}/k_{g2}$

$c_3 = \{ k_p J b_2 - B [B\beta + k_v J(\beta - 1)] \}/k_{g2}$

$d_1 = \{ -[k_p T^2 + 2k_v T + 3] \}/k_{g1}$

$d_2 = \{ k_v T + 3 \}/k_{g1}$

$$d_3 = -1/k_{g1}$$

The root locus plot for the modified motor torque equation for joint 1 is plotted in Figure 6-1 using crosses to mark the root locations. The original inertia sensitivity from Figure 5-3 is plotted for purposes of comparison, using circles. It can be seen that the root variations from the modified motor torque equation are concentrated within a small area, while the variations from the original motor torque equation cover the entire plot.

B. Reduction of Sampling Rate Sensitivity

The sensitivity of the present arm control system to variations in sampling rate was demonstrated in Section V-C. It was shown that, when the feedback gains are held fixed, all of the joints become unstable between 5 and 35 hz.

Ideally, we would like to keep the closed loop poles in stationary positions as the sampling rate is varied. Looking at the z-plane poles can be misleading when the sampling rate is varied, however, because the same z-plane poles give different responses at different sampling rates. The effect of sampling rate variations on the arm's dynamic response is best illustrated by using a transformation to map the z-plane poles to the s-plane.

Keeping the s-plane poles in stationary positions as the sampling rate is varied minimizes the effect of sampling rate on the arm's speed of response. To determine how the feedback gains would have to be changed to keep the s-plane poles stationary in the presence of sampling rate variations, a transformation was used to determine how the z-plane characteristic equation would have to. vary to keep the s-plane poles constant. The sampling rate was then varied and the gains required to keep the s-plane poles stationary were computed using equation (3-15). It was found that the required proportional and derivative feedback gains doubled as the sampling rate was doubled. The integral feedback' gains varied only slightly as the sampling rate was varied. By modifying the motor torque equation so that the proportional and derivative feedback terms are divided by the sampling period, the movement of the s-plane poles is significantly reduced. Thus, the feedback portion of the motor torque equation given in equation (6-2) should be modified to

$$T(k)_{fb} = -k_c J(k) \{ k_p [\theta(k) - \theta_c(k)]/T + k_v [w(k) - \omega_c(k)]/T + k_i \sum_{j=1}^{k} [\theta(j) - B_r(j) I] \} \qquad (6\text{-}4)$$

to become less dependent on sampling rate and inertia.

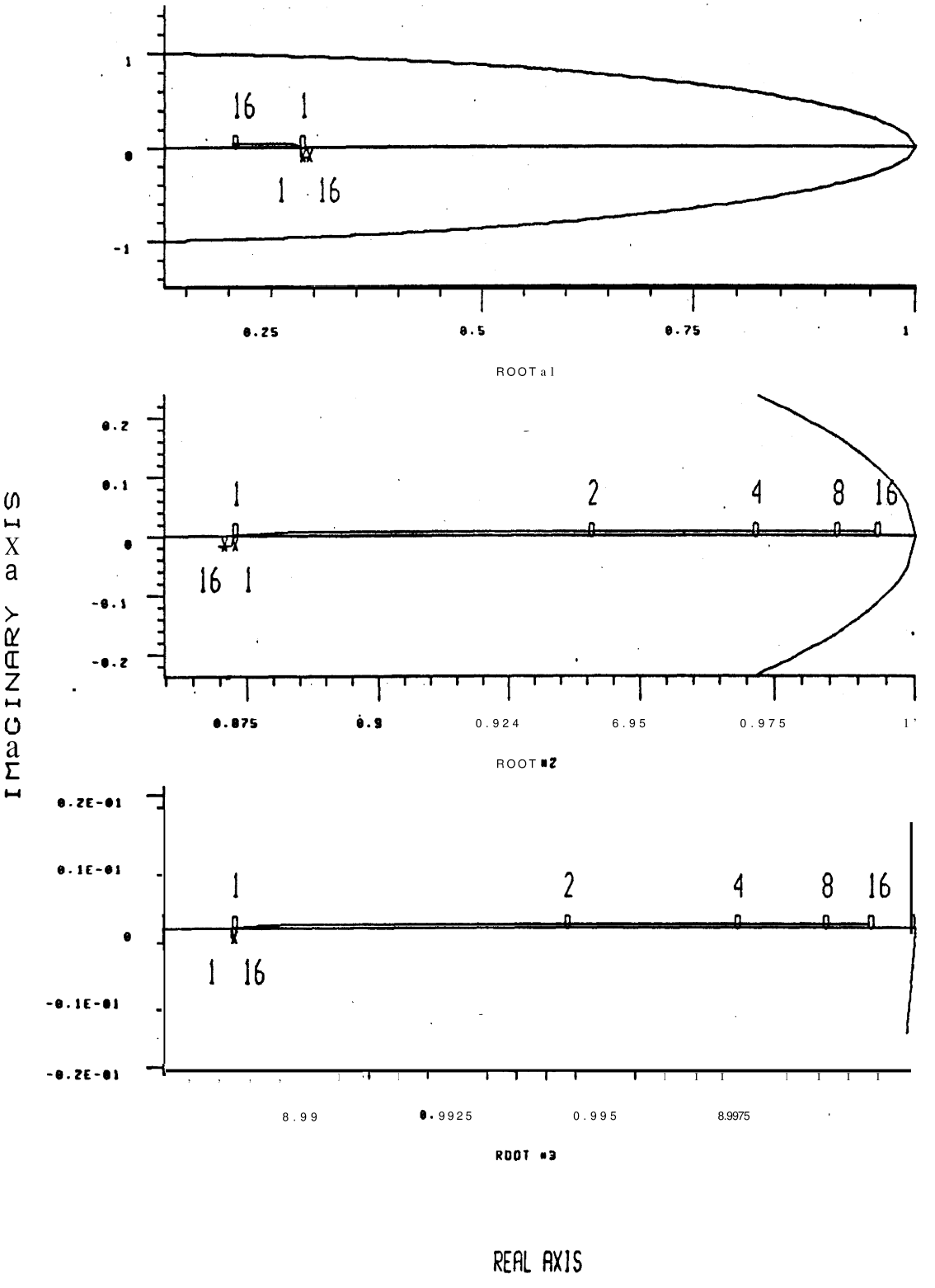


Figure **6-1:** Reduced Effect of Inertia - Joint **#1** Root Loci

With the recommended motor torque equation (6-4), the final joint transfer function becomes

$$H(z) = \frac{\theta_a(z)}{\theta_c(z)} = \frac{k_c k_g [b_1 z - b_2][z^3 + d_1 z^2 + d_2 z + d_3]}{(Tz)^2 [z^3 + c_1 z^2 + c_2 z + c_3]} \qquad (6\text{-}5)$$

where

k_g = gain constant = k_{g1}/k_{g2}

$k_{g1} = k_p T + k_i T^2 + k_v + 1$

$k_{g2} = B^2$

$b_1 = TB + J(\beta - 1)$

$b_2 = TB\beta + J(\beta - 1)$

$c_1 = \{ b_1 J [k_i + k_p/T] - B [B(2 + \beta) + k_v J(\beta - 1)/T] \}/k_{g2}$

$c_2 = \{ B [B + 2B\beta + 2k_v J(\beta - 1)/T] - b_2 J [k_i + k_p/T] - k_p J b_1/T \}/k_{g2}$

$c_3 = \{ k_p J b_2/T - B [B\beta + k_v J(\beta - 1)/T] \}/k_{g2}$

$d_1 = \{ - [k_p T + 2k_v + 3] \}/k_{g1}$

$d_2 = \{ kv + 3 \}/k_{g1}$

$d_3 = -1/k_{g1}$

The effect of sampling rate variations on the new motor torque equation described above is illustrated in Figure 6-2 where the closed loop poles are plotted in the z-plane. By comparing Figures 5-3 and 6-2, it can be seen that the sampling rates at which the pates cross the unit circle and cause the joints to become unstable are lower in every case for the modified motor torque equation.

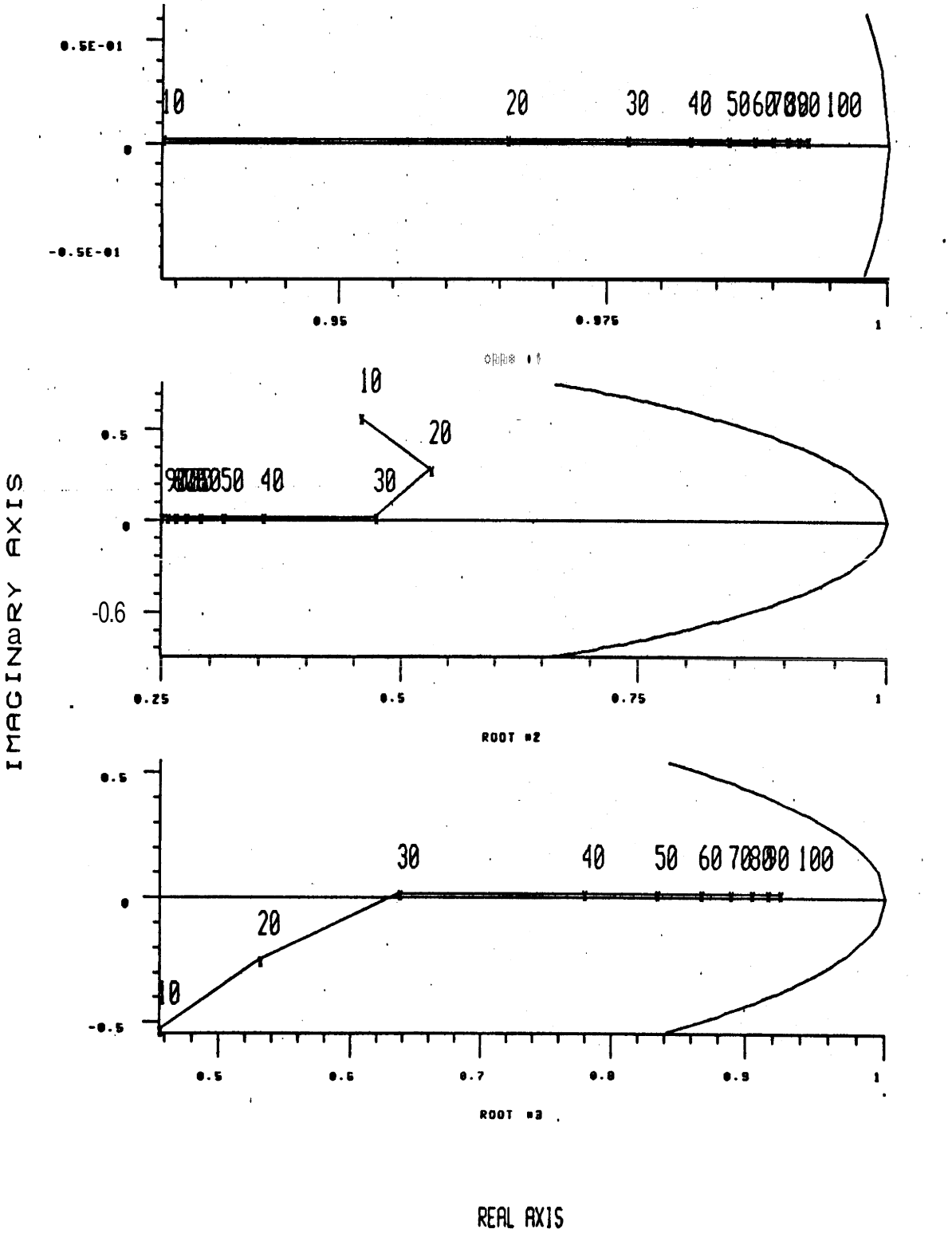


Figure 6-2: Reduced Sampling Rate Sensitivity . Joint #1 Root Loci

The modifications to the motor torque equation could be carried one step farther by including equation (3-15) for computing the feedback gains in the motor torque equation. The resulting equation would consume a great deal of computation time on the computer, but theoretically, the s-plane poles could be made perfectly stationary in the presence of sampling rate variations. Thus, the speed of response of the arm could be held fixed regardless of the sampling rate. This sounds somewhat idealistic, and it is. Whenever the sampling rate is reduced, the arm perfomance is degraded in several ways.

1. The arm's response time increases (even when the s-plane poles are held stationary).

2. The arm's sensitivity to disturbances (such as those caused by friction) increases.

3. The arm's sensitivity to parameter variations (such as those caused by errors in the estimation of inertia) increases.

4. The roughness of the arm motions increases because the steps begin to appear in the command signal from the digital to analog converter.

The above factors are discussed in detail in reference [KA]. Before the effects of sampling rate on the arm's response can be fully understood, the relative importance of each of the above factors must be determined.

The effect of the sampling rate on the response time of the joints is illustrated in Figure 6-3 for joint 1. For the motion shown in Figure 6-3, joint 1 was commanded to move 90 degrees in one second. It can be seen that the response time increases as the sampling rate is decreased, although the response time does not increase significantly until the sampling rate is reduced to 20 hz.

The sensitivity of the joint to disturbances is shown in Figure 6-4 where joint 1 was again commanded to move 90 degrees in one second. To simulate a disturbance, the coulomb friction compensation was removed from the motor torque equation. It can be seen that the disturbance creates an additional 'error of almost 0.9 degrees in the 20 hz plots, but the error for the 62.5 hz plots never exceeds 0.2 degrees.

The joint's sensitivity to parameter variations was simulated by altering the computed inertia. The resulting error is plotted in Figure 6-5. It can be seen that the additional error generated by varying the inertia is worse for the 20 hz sampling rate than for the 62.5 hz rate.

The roughness due to the steps in the digital to analog converter signal is not an important factor. At reasonable sampling rates above 20 hz, the roughness cannot be seen in the position plots in Figure 6-2.

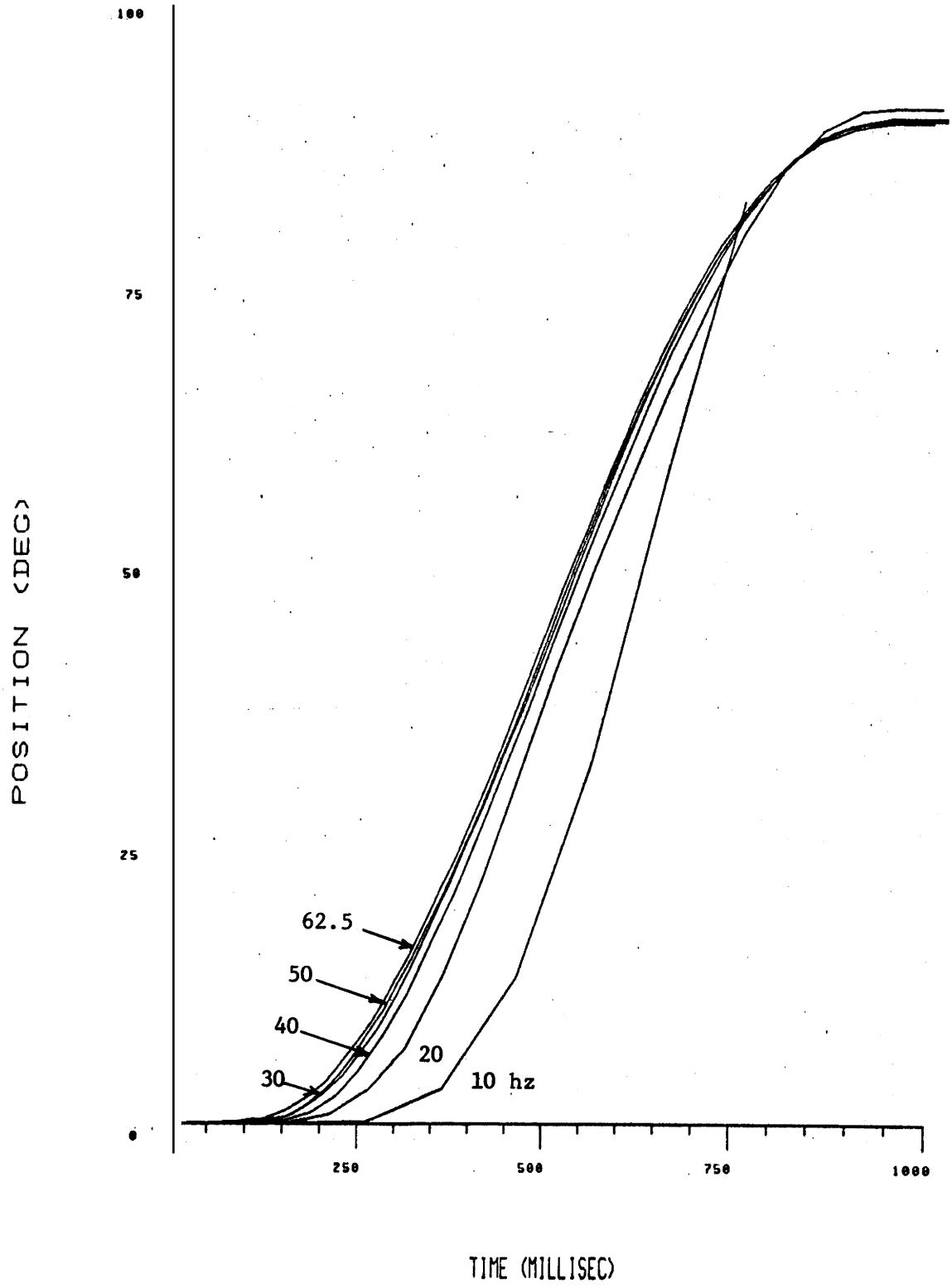


Figure 6-3: Effect of Sampling Rate on Response Time - Joint #1

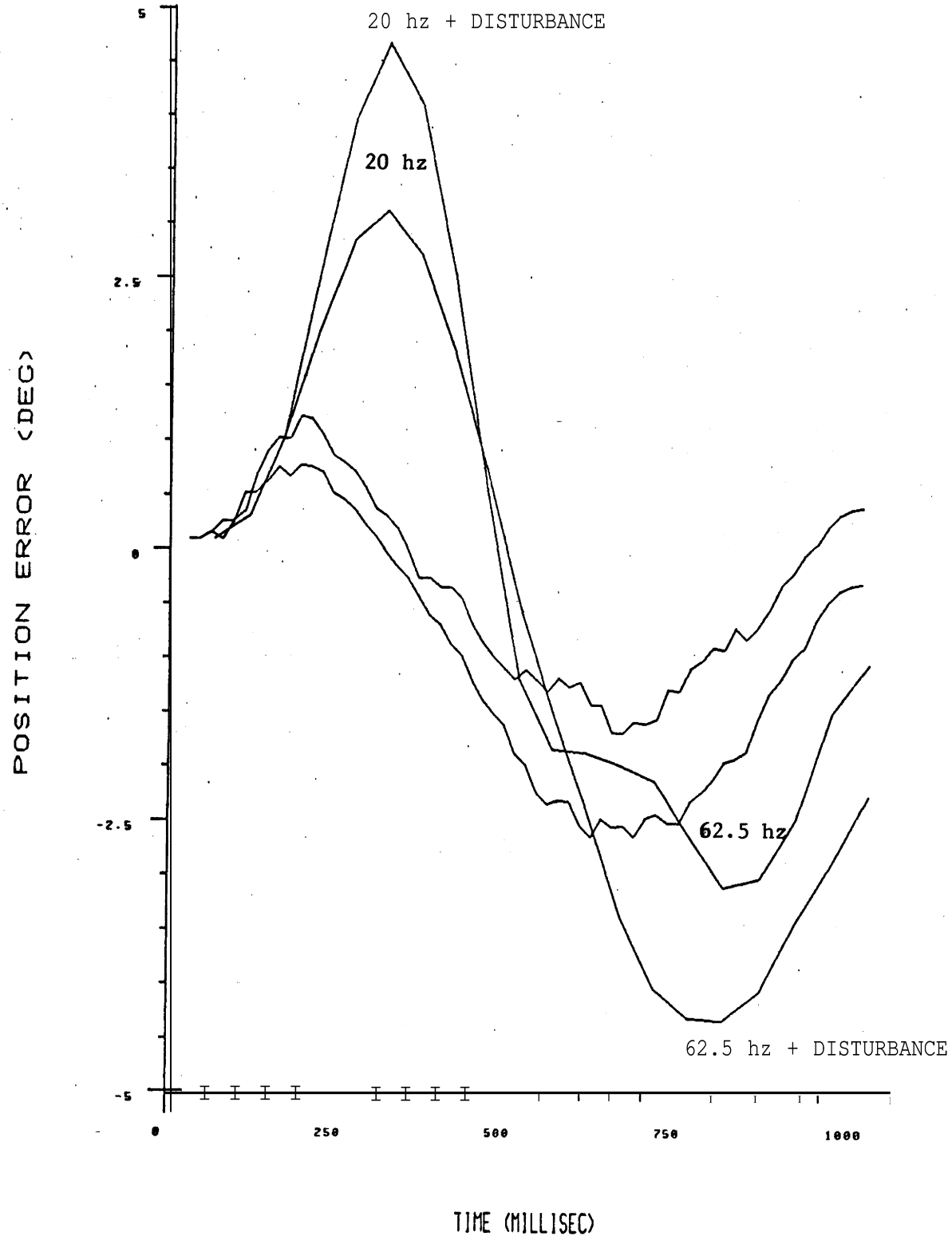


Figure 6-4: Effect of Sampling Rate on Sensitivity to Disturbances - Joint #1

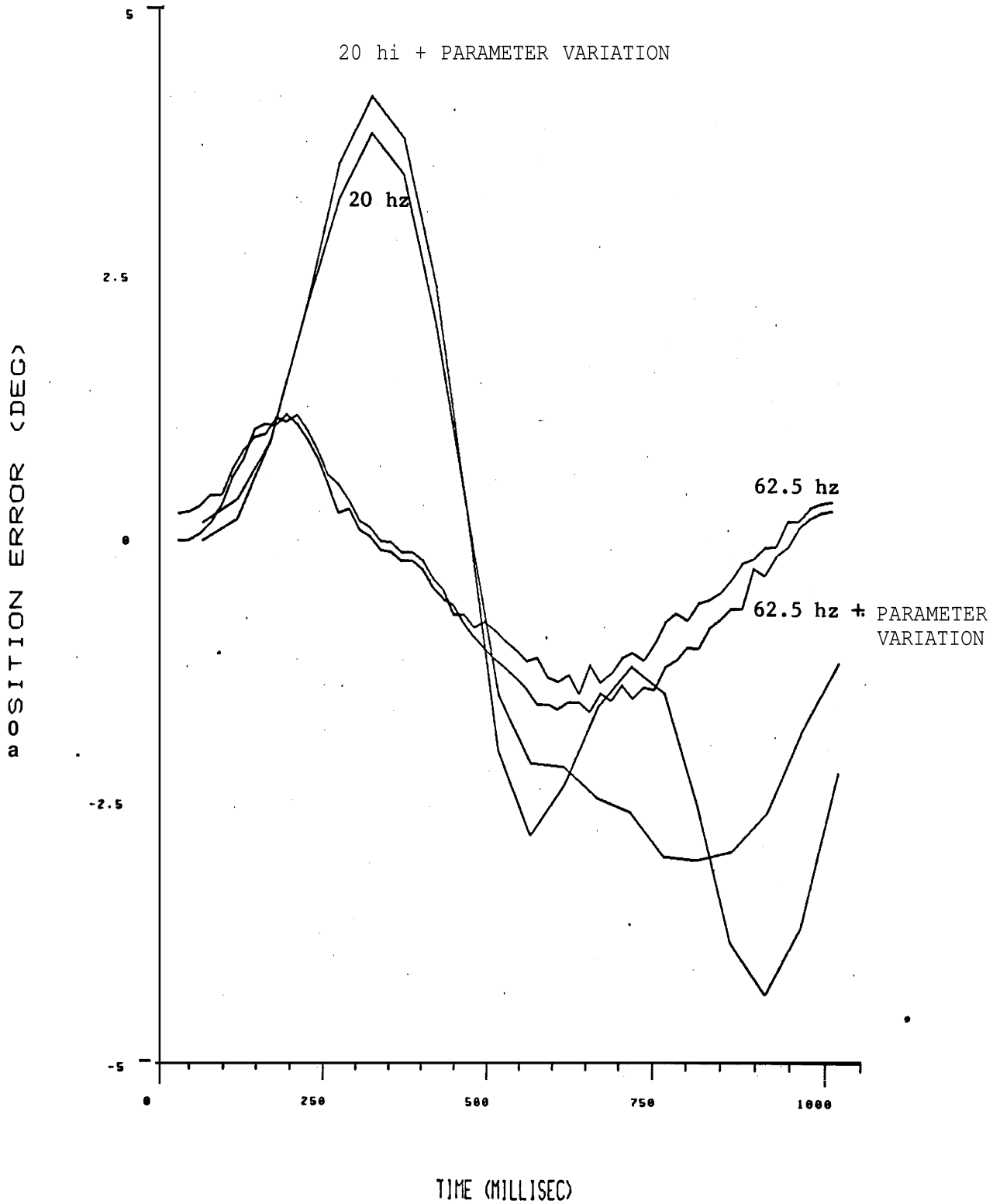


Figure 6-5: Effect of Sampling Rate on Sensitivity to Parameter Variations - Joint #1

In Figure 6-6, the error appearing in joint 1 for a one second, 90 degree motion is plotted as a function of the sampling rate. Here the feedback gains have been adjusted so that the s-plane poles remain fixed when the sampling rate is changed. It can be seen that the additional error' created by reducing the sampling rate is almost insignificant at 40 hz. At 62.5 hz the maximum error in the middle of the motion is 1.5 degrees, while at 40 hz, the maximum error is only 2.0 degrees.

The root mean square (RMS) error defined by equation (6-6) has been calculated for the motions described above and plotted in Figure 6-7A. Surprisingly, the rms error seems to have no correlation with the inertia of the joints. In Figure 6-7B, the plot of rms error has been normalized by subtracting the 62.5 hz value from all of the rms errors and then dividing by the 10 hz value. The scale of the normalized rms error has been expanded in Figure 6-7C. The normalized rms error gives a measure of the relative sensitivity of the joints to sampling rate. It can be seen that joint 2 has the least percentage increase in rms error when the sampling rate is reduced to 40 hz. Joint 6 has the largest increase.

$$E_{rms} = [\sum_{i-1}^{n} (T e_i)^2]^{1/2} \qquad (6\text{-}6)$$

The final recommendation of the sampling rates for each of the joints is an engineering judgement based on the accuracy required in the middle segments of a motions and on the amount'of processing time available for control functions. The best performance will always be obtained at high sampling rates. When operating conditions limit the arm sampling rate, the joints should be sampled at rates which are based on the relative sensitivity of the joints. For the Stanford Arm, I recommend that the available sampling time be distributed as shown in Table 6-1. This table also gives the recommended minimum sampling rate for each joint. When the joints are operated below these sampling. rates, the increased error and roughness will soon become noticeable to the eye.

JOINT	% OFSAMPLING TIME	MINIMUM SAMPLING RATE (hz)
1	17.5	47
2	14.9	40
3	17.8	48
4	15.2	41
5	17.1	46
6	17.5	47

Table 6-1: Sampling Rate Recommendations

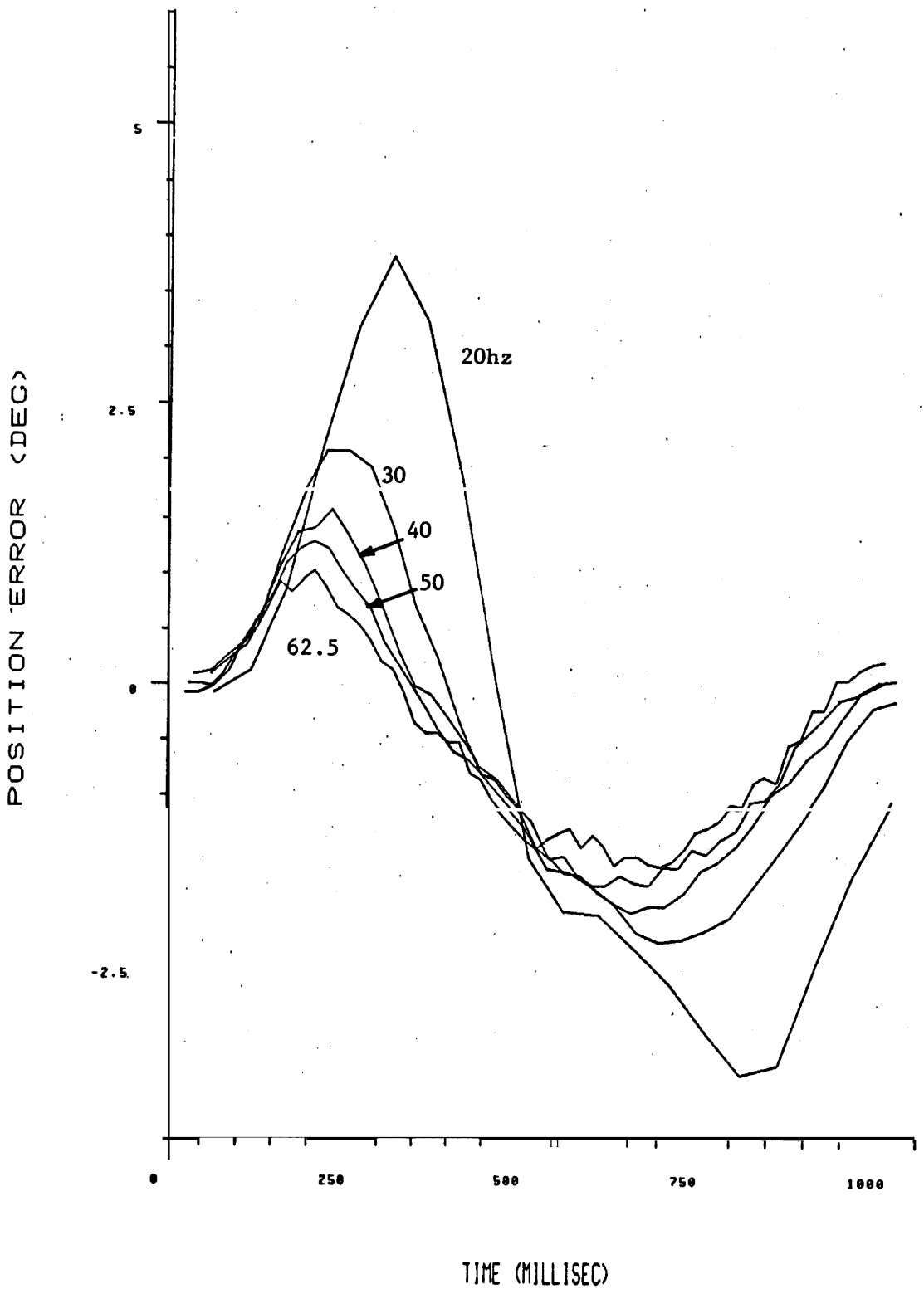


Figure 6-6: Effect of Sampling Rate on Joint #1 Error

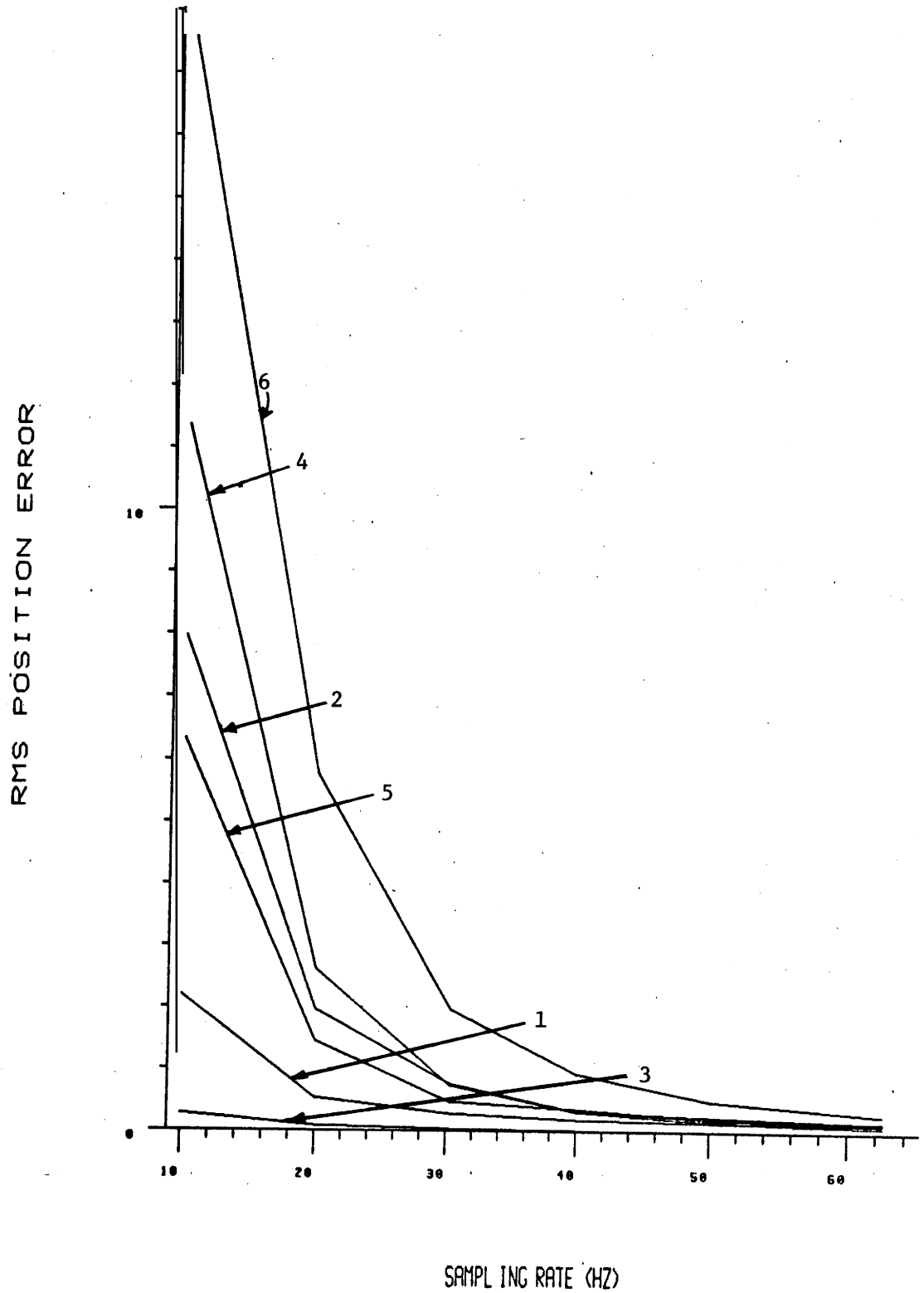


Figure 6-7A: Effect of Sampling Rate on RMS Error

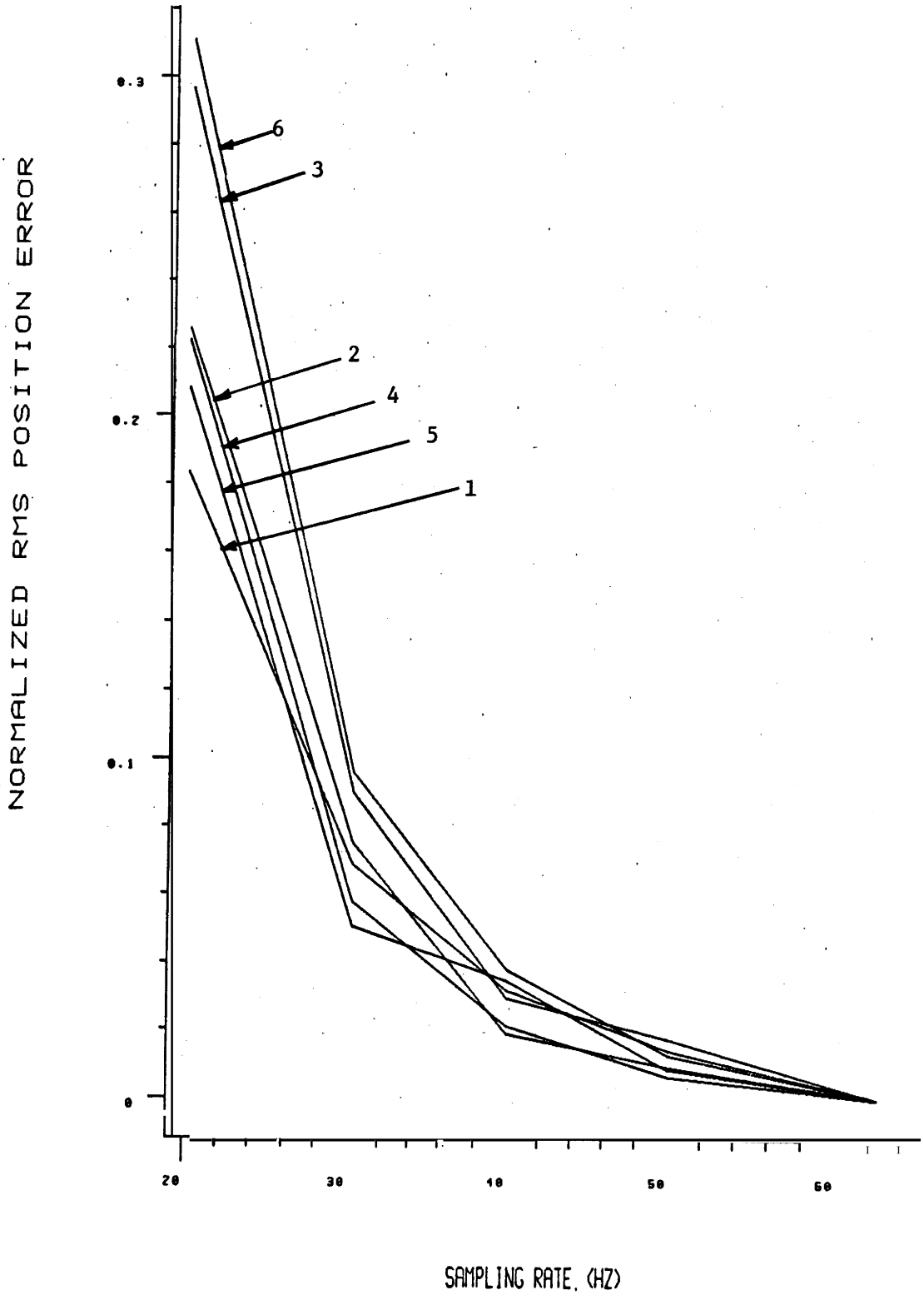


Figure 6-7B: Effect of Sampling Rate on Normalized RMS Error

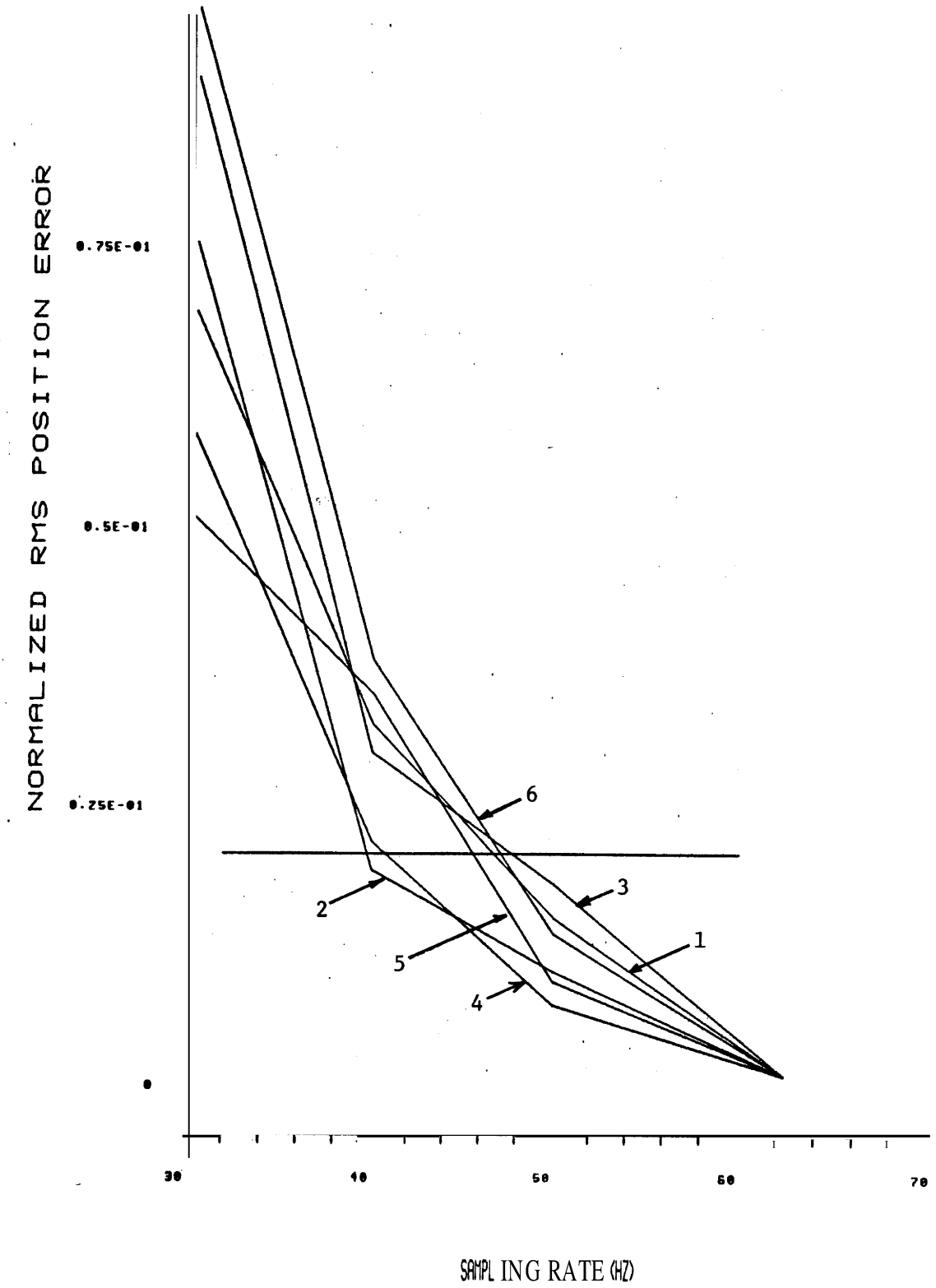


Figure 6-7C: Effect of Sampling Rate on Normalized RMS Error - Expanded Scale

VII. CONCLUSIONS

The discrete z transform transfer function was computed for the joints of the Stanford robot arm using both a classical model and a state space model. Several measurement techniques were then devised for measuring and tabulating the parameters of the transfer function for each joint.

The sensitivity of the present control system to variations in inertia and sampling rate was demonstrated and recommendations were made to reduce these sensitivities. The suggestion was made that the feedback gains should be adjusted to maintain the same s-plane pole locations whenever the sampling rate is changed.

It was shown that the sampling rate of the joints can be decreased at the expense of reduced speed of response, increased sensitivity to disturbances and to parameter variations, and increased roughness due to the larger discrete steps in the digital to analog converter output signal. It was noted that the additional roughness and reduced response speed were not significant at reasonable sampling frequencies. The increased sensitivity to disturbances and to parameter variations was significant and was the limiting factor governing the minimum effective sampling rate. Plots were made showing the error in each of the joints as a function of sampling rate. From these plots, recommendations were made concerning the relative sensitivity of each of the joints to sampling rate and the minimum effective sampling rate of each joint;

VIII. REFERENCES

[BE] Bejczy, A.K., *Robot Arm Dynamics and Control,* Technical Memorandum 33-669, pp. 48-64, Jet Propulsion Laboratory, Pasadena, Ca., February 15,1974.

[CA] Cadzow, J.A. and H.R. Martens, *Discrete-Time and Computer Control Systems,* Prentice Hall, Inc., Englewood Cliffs, N. J., pp. 55-57 and pp. 100- 104, 1970.

[CH] Chen, C.T., *Introduction to Linear System Theory,* Holt, Rinehart & Winston, Inc., New York, 1970.

[FR] Franklin, C.F. and J.D. Powell, *Digital Control,* Notes prepared for EE207, Stanford . University, Stanford, Ca., Winter 1976.

[GO] Gopinath, B., *On the Control of Linear Multiple Input-Output Systems,'* Bell System Technical Journal, vol. 50, pp. 1063-1081, March 1971.

[KA] Katz, P., *Selection of Sampling Rate for Digital Control of Aircrafts,* SUDAAR Report No. 486, Stanford University, Dept. of Aeronautics and Astronautics, Stanford, Ca., September 1974.

[OG] Ogata, K., *Modern Control Engineering,* Prentice Hall, Inc., Englewood Cliffs, N.J., pp. 228-239 and pp. 384-387, 1970.

[RO] Roderick, M. D., *Discrete Control of a Robot Arm,* Engineers Thesis, Electrical Engineering Department, Stanford University, Stanford, Ca., July 1976.

VIII. POINTY USER MANUAL

M. Shahid Mujtoba

Artificial Intelligence Laboratory
Computer Science Department
Stanford University

The author is a graduate student in the Industrial Engineering Department,

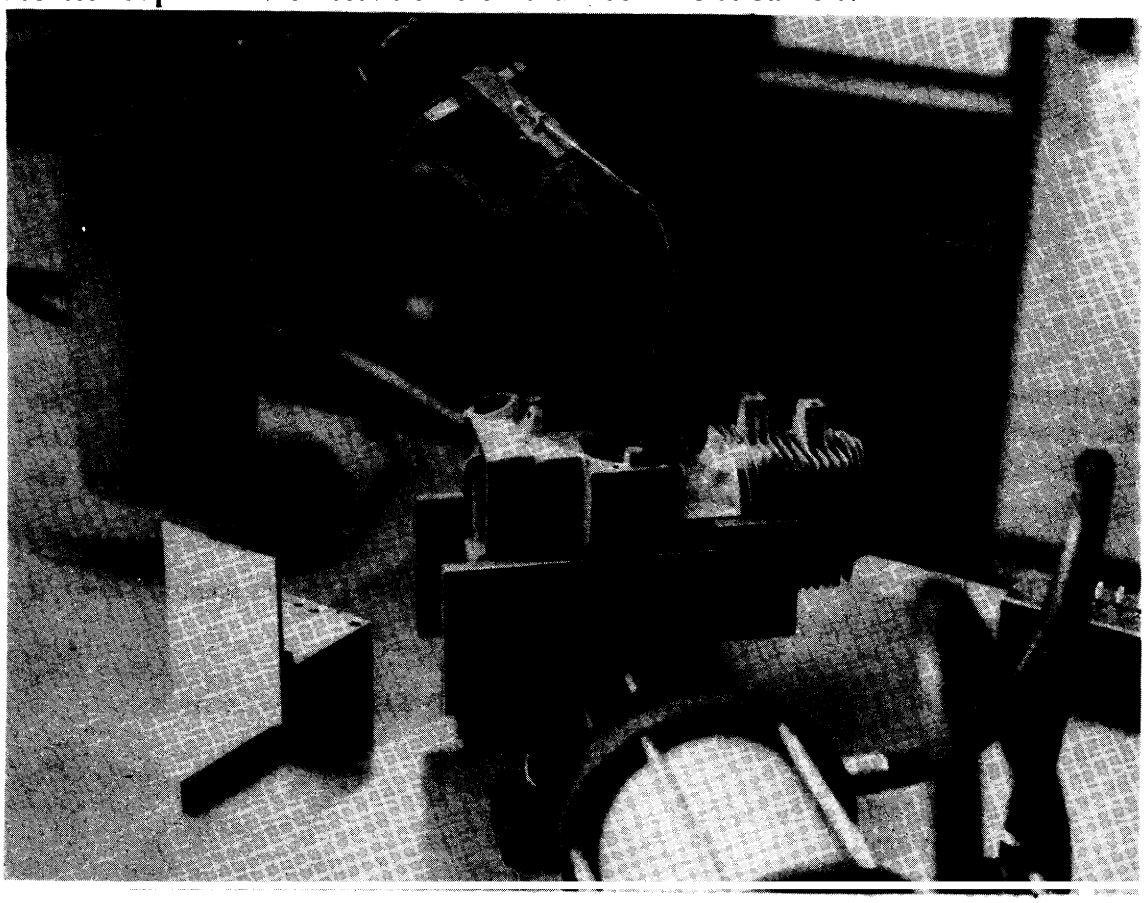
AN INTERACTIVE SESSION USING POINTY

POINTY is a program that helps to generate an affixment structure of frames as described in the AL manual [AI Memo-243, *AL, A Programming System for Automation,* by Raphael Finkel, Russell Taylor, Robert Bolles, Richard Paul and Jerome Feldman, November 1974]. The user is relieved of two burdens:

(1) The tedium of measuring the locations of workspace features in three dimensions with a ruler and protractor by simply pointing to those locations with the manipulator.

(2) The mental gymnastics involved in determining the frames and transes from the physical measurements by using POINTY as a sort of desk calculator.

POINTY was designed by Russell Taylor and David Grossman and is described in AI Memo-274,, *Interactive Generation of Object Models with a Manipulator,* December 1975, and has been implemented on both the Yellow and Blue Arms at Stanford.



At this stage POINTY is used to generate affixment structures of the world model used in AL programs. Since AL has been implemented on the Blue Arm, the following directions are for use of POINTY on the Blue Arm. (The Yellow Arm runs on the PDP-IO under

WAVE.) In the explanations, bold type (e.g. R11TTY) represents characters typed by the user, while italicized type (e.g. CORE) represent response by the system.

After logging in at a III or a DD terminal, the first thing to do is to load the PDP-11 with the servo program to read joint angles, etc. using the following sequence of commands:

.R 11TTY <CR>

This loads 11TTY which is a program that loads other programs into the PDP-11. 11TTY when loaded responds with an asterisk for further instructions.

*ZERO CORE [CONFIRM]<CR>
*GET SAV FILE - DIAG[HAL,HE]<CR>
*START AT (D FOR DDT) - D<CR>

On the VT05 (the DIGITAL terminal with the tinted glass in front of the screen) type

W <ALT> G

You should see continuous scanning of the VT05 screen as follows:

JT 1	JT2	JT 3	JT 4	JT 5	JT 6		HAND
'179.99	-89.86	13.99	-89.82	89.86	.08		1.99
-1126	-61	965	627	-1241	-35		-159

X	.Y	Z	0	,A	T
43.57	16.87	10.89	89.91	89.64	.00

. JT 1 through 6 except 3 represent joint angles in degrees. JT 3 gives the reading of the boom extension in inches; the hand opening is given in inches. The second row of numbers represent the A/D readings. X Y Z 0 A T represent the x,y,z coordinates and the orientation of the hand.

If at any time you accidentally hit one of the other keys of the VT05 and scanning stops, it can be started again by the W<ALT> G sequence.

If the VT05 does not respond as described, do the following at the PDP-11 console:

- Press HALT
  Set Switch Register to octal 0
  Press LOAD ADO
  Set Switch Register to octal 1
  Press DE?
  Press RUN
  Go back to VT05 and do the W<alt>G sequence again

When the VT05 is running along happily, 1 ITTY must be killed by hitting the CALL key on the terminal. Then type

.RU POINTY[HAL,HE]

System will respond with (in the following italics represent response of the system)

BAIL is your command scanner.
BAIL ver. 6- Jun-76 using POINTY.BAI[HAL,HE]
End of BAIL initialization.
4207 ***OUTSTR("BAIL** is* your *command scanner?);*
;BAIL;
1:

POINTY is now ready to accept commands through BAIL. Release all the brakes of the arm and move the hand to the reference point (called fiducial point). Grab the fiducial point between the fingers, then reset the brakes. The hand co-ordinates will be shown on the VT05. We are ready to give the first command to POINTY

DEFFID;<CR>

Note that the semi-colon must be typed in. This instruction will define the position of the fiducial point in world coordinates. Note that the co-ordinates of ARM and FIDUCIAL are the same on the table. The last three co-ordinates of the transform represent the location of the hand co-ordinates while the first three represent the orientation information in degrees (0 A T X Y Z). Do not move the arm until the system responds with

1:

Release all the brakes again and grab the pointer in the hand, and reposition the hand so that the tip of the pointer is now pointing to the fiducial point.

ATFID;<CR>

This defines the relationship between the pointer and the arm in terms of the relative position between them. Note that the TR of POINTER is no longer (0,0,0,0,0,0). The transform of POINTER is with respect to arm. If POINTER were made independent of the affixment structure at this point, its co-ordinates would be those of FIDUCIAL. To verify this do the following:

1: APUSH(ABSLOC("POINTER"));<CR>

This pushes the absolute value of the pointer on the arithmetic stack "A:", which is the default arithmetic stack at initialization. To select stack "B:" instead you could have done instead:

1: APUSH(ABSLOC("POINTER"),"B:");<CR>

Verify that this value is the same as that of FIDUCIAL on the display screen.

You should note that merely moving the arm does not update the value of ARM in the affixment structure, until an explicit instruction has been given to do so. The routine that does this is READARM; and can be called directly by you. Certain other instructions (like POINTIT; and GRABBIT; described below) also call READARM, so in those cases you need not call it explicitly.

Let us now find the location of the base of the pneumatic vise, and the coordinates of the jaws with respect to the base. First, we will arbitrarily choose the corner of the base plate closest to the top left hand corner of the table as we face the blue arm as the origin of our co-ordinate system translated in world co-ordinates without any rotation. Release the brakes again and point the end of the pointer to the base point of the base plate, making sure that the pointer does not bend or deform in the process.

```
POINTIT;<CR>
```

The Transformation of the base plate origin appears on the A: stack. This instruction is equivalent to the two instructions

```
READARM;<CR>
APUSH( ABSLOC( "POINTER"));<CR>
```

We could have done POINTIT("A:"); or POINTIT("B:"); to put the frames into the appropriate arithmetic stack. Note that the orientation of the vectors are non-zero. Let us edit the values so that they are zero, since we want the origin of the base plate to be merely translated without being rotated. We know that the pointer is pointing to the origin of the base plate. Let us define a new node called "BASE-PLATE".

```
1:
    MK_NODE( "BASE_PLATE");<CR>
```

Push the transform of the pointer on top of the B: stack.

```
1:
    APUSH( ABSLOC( "POINTER"),"B:");<CR>
```

Note that there are two arithmetic stacks A: and B: and the default stack is the last used stack; initialization makes the A: stack the default stack initially.

```
1:
```

Let us change the orientation of the value at the pointer by changing the value of the top element of the B: stack

```
TEDIT;<CR>
```

Computer responds with

```
1:
    APUSH(TR(156,132,-102,-.030,49.3,.20),"B:");
```

Edit the first three values to make them zeros

```
1:
    APUSH( TR( 0, 0, 0, -.030,4 9.3,. 20),"B:");<CR>
```

The new value will appear on top of the B: stack. We want this value to be the value of BASE-PLATE;

```
1:
    ABSSET;<CR>
1:
```

Let us now define a point on the vise, say the outer jaw of the vise. The z-axis points in the direction opposite to the world co-ordinate z-axis, and the x-axis is 45 deg from world coordinates.

Point the pointer to the corner of the outer jaw.

```
POINTIT;<CR>
```

1:

Now point the pointer to a point vertically below the previous point - this is equivalent to pointing to a point on the z-axis.

. POINTIT;<CR>

1:

Point the pointer to a point on the face of the vise (a point on the x-z plane).

POINTIT;<CR>

1:

Using the x,y,z position information of the last three transforms (ignoring the 0 A T values) construct a transform giving the location and orientation of the outer jaw.

CONSTRUCT;<CR>

1:

Define a new node "OUTER-JAW"

MK_NODE("OUTER-J AW");<CR>

1:

ABSSET("OUTER_JAW");<CR>

This sets the value of the transform on the arithmetic stack as the absolute location of OUTER_JAW. We know that OUTER-JAW is fixed rigidly to BASE-PLATE, so let us define it as such. Set "BASE-PLATE" on the "D:" stack (the DAD stack).

1:

CPUSH(λ("BASE_PLATE"),"D:");<CR>

1:

RIGID;<CR>

Note the asterisk which marks OUTER--JAW as rigidly connected to BASE-PLATE. A "+" indicates non-rigid affixment, while a "-" indicates independent affixment. Be very careful of rigid affixments - when one of the members of a rigid affixment is changed, the other is affected too. In the above example, had OUTER-JAW been rigidly affixed to BASE-PLATE before the ABSSET instruction, the execution of the latter would have changed the value of BASE-PLATE since the affixment structure would have updated BASE-PLATE on the basis of the relative transform set up when RIGID was invoked.

1:

Let us now define another point at the other end of the jaw; by measurement, we find that it is 8 inches along the x axis and 0.5 inches along the z-axis of "OUTER-JAW"; First we define the transformation we want

APUSH(TR(0, 0, 0, 8, 0,0.5),"B:");<CR>

1:

Define a new node called "OUTER_JAW2"

MK_NODE("OUTER-JAW 2");<CR>

1:

GOSON("D:");<CR>

Cursor D: is now at OUTER-JAW

1:

RIGID;<CR>

This connects OUTER-JAW2 RIGIDLY to OUTER-JAW. However, the transform assumes that OUTER-JAW2 was at the origin. Let us set the value of the top of the B:

stack as the relative location of OUTER_JAW2.

1:

RELSET("OUTER-J AW 2");<CR>

We now have a lot of garbage on the Arithmetic Stacks; let us get rid of them, starting with stack B:

1:

APOP;<CR>

This pops the top element off the stack. Keep on doing this until there are no more elements on the B: stack.

1:

Now start emptying the A:stack

APOP("A:");<CR>

After this keep on doing APOP; if at any time just after popping you decide you really want the value, type OOPS; and the value will be retrieved; however, if instead of APOP you say AFLUSH; you won't be able to get the value again by saying OOPS;

1:

Let us now define the center of the outer jaw and call it "OUTER-JAW-C" and join' it rigidly to BASE-PLATE. The following operations will do the trick.

MK_NODE("OUTER-J AW_C");<CR>

1:

APUSH(TR(0,0,0,4,0,.25),"A:");<CR>

1:

RIGID;<CR>

1:

RELSET;<CR>

1:

GODAD("D:");<CR>

1:

RIGID;<CR>

This has been done by rigidly affixing OUTER-JAW-C to OUTER-JAW and defining its relative position and then reaffixing it rigidly to BASE-PLATE.

Having done enough editing for one day, let us save the model in an AL-FILE and in a P-FILE. An AL-FILE will contain the model of the affixment in terms of AL declarations for future use with AL programs. A P-FILE will contain instructions to generate a model which POINTY understands. Note that POINTY' cannot understand the affixment structure in terms of AL declarations. It can only understand the type of instructions we have been using here.

To clean things up before saving what we want, let us move the "D:" pointer to WORLD, and "N:" pointer to BASE-PLATE. The final cursor of "N:" and "D:" referenced before saving will point to the node to be saved.

1:

GODAD;<CR>

repeatedly until D: points to "WORLD". Then do

1:

GODAD("N:");<CR>

and if necessary GODAD; repeatedly until N: points to BASE-PLATE

1:

NONRIGID;<CR>

This will affix BASE-PLATE non-rigidly to the world.

1:

AL_WRITE<CR>

Computer responds with

OUTPUT FILE (NULL TO FORGET IT)=

Type in desired filename (say VISE.AL). Note that this filename will appear on the display. The file remains open for future input to it unless it is explicitly closed as in the following instruction, or until EXIT; is typed. Other nodes can be dumped in AL declaration format by moving the "N:" or "D:" cursor to the relevant node before typing AL-WRITE;. Note that only one AL_FILE can be open at any one time. If an AL-FILE is open the computer will not respond with the *OUTPUT FILE (NULL TO FORGET IT)* = prompt. If AL declarations are to be saved in more than one file, the files have to be opened and closed one at a time.

1:

AL_CLOSE;<CR>

This instruction has the effect of closing the output file. Computer responds with

CLOSING VISE-AL

1:

PSAVE;<CR>

Computer responds with

OUTPUT FILE (NULL TO FORGET IT)=,

Type in desired filename (let's call it VISE.P). Instructions to generate the affixment structure connected to the node pointed to by "N:" cursor will be dumped out.

1:

Let's call it a day and quit.

EXIT;<CR>

End of SAIL execution

Note that this instruction automatically closes any files that are open. If <CONTROL>C or CALL had been hit on the keyboard, any files would be lost, so be careful if you do not want to lose your data. Note that if N: had been pointing to WORLD before we had asked for PSAVE or AL-WRITE, everything would have been saved, including the ARM, FIDUCIAL and POINTER transformations, which we are not really interested in.

The last instruction that we typed in gives us back to the monitor again, so let us look at the AL_FILE and the P-FILE that we have generated. To do this we type

.ETV VISE.AL<CR>

The monitor will respond with

NEED TO REFORMAT VISE.AL.OK?(Y OR N)

to which you should respond Y.

Examination of VISE.AL will show that it consists of FRAME Declarations and transformations and affixment relations of OUTER-JAW, OUTER-JAW-Z, and OUTER-JAW-C and BASE-PLATE. Examination of VISE.P in a similar manner will show a set of POINTY instructions. You can use the text editor to modify or add instructions if you like, so long as you make sure the arguments are in the right places.

Suppose we want to continue and start over again the next day. Go through the whole process from the beginning up to and including RU POINTY, and wait till POINTY, after initializing, prompts:

1:

DSKIN("VISE.P");<CR>

POINTY will then read in the state of the world as we read it in previously into VISE.P; affixment structures for BASE-PLATE with OUTER-JAW, OUTER-JA W2, and OUTER-JAW-C will then appear. We now redefine the positions of the FIDUCIAL and POINTER with instructions similar to what we used before.

1:

DEFFID;<CR>

1:

ATFID;<CR>

1:

The BASE-PLATE location has shifted since the last time we used it, so let's redefine the location. (Looking at the location, we see that it has shifted 4.5 inches in the -y direction.) Release the brakes, and point the pointer to the origin.

POINTIT;<CR>

1:

Move the pointer to a point on the Z-axis and then type POINTIT, and then. move the pointer to a point in the XZ-plane and type POINTIT; again.

POINTIT;<CR>

1:

POINTIT;<CR> .

1:

Let's now construct the trans of the origin.

CONSTRUCT;<CR>

1:

Let's now define the position of the screwdriver. It is sitting on the BASE-PLATE, and by grabbing it, we want to be able to define its position. Release the brakes again, and move the arm to the screwdriver and grab the screwdriver between the fingers.

GRABBIT;<CR>

-J:

MK_NODE("DRIVER");<CR>

1:

ABSSET("DRIVER");<CR>

An alternate way of doing the same thing without releasing the brakes is to go through the following sequence

1: FREE;<CR>

This frees the joints and enables you to move the arm around for five seconds - after moving the arm around to the location you want it, you should press the BLACK BUTTON on, the control box to apply the brakes. This feature is not supported yet.

```
1 :     HERE("DRIVER");<CR>
```

Let's try it but without the FREE, and call the new position DRIVER2

```
1:
        HERE("DRIVER2");<CR>
```

Note that the values of DRIVER and DRIVER2 are the same. It's uncomfortable to have two of the same node around, so let's kill DRIVER. First we have to get to DRIVER. Note that DRIVER is the elder brother of DRIVER2 (just below it), so we have to go there first.

```
1:
        ELDER;<CR>
```

Cuisor N: has now shifted to DRIVER;

```
1:
        KILL;<CR>
```

This kill's node DRIVER. Now we have a node DRIVER2 without DRIVER, so let us rename DRIVER2 as DRIVER, first making sure that "N:" is pointing to DRIVER2.

```
1:
        NAME_NODE( "DRIVER");<CR>
```

We know that "DRIVER" is non-rigidly fixed to BASE-PLATE, and we want to show this on the 'affixment structure.

```
1:
        CPUSH(λ("BASE_PLATE"),"D:");<CR>
1 :
        NONRIGID;<CR>
```

DRIVER is now the youngest son of BASE-PLATE. Suppose we now want to put "N:" at FIDUCIAL. One way to do it would be to do a GODAD; followed by ELDER; Another way, if we want to do this pretty often is to define a MACRO (call it UNCLE).

```
1:
        MDEF("UNCLE");<CR>
```

Computer responds with

TYPE IN MACRO **BODY.(<ALT> WHEN** *DONE):*

So we type in

```
        GODAD;    ELDER;<ALT>
            -UNCLE DEFINED.
```

```
1:
```

Let us call UNCLE to verify that it works.

```
        MCALL(          "UNCLE");<CR>
```

"N:" jumps to FIDUCIAL, thus verifying that UNCLE works. Let us redefine the macro so that UNCLE means to go to younger brother rather than elder brother.

```
1:
        MDEF;<CR>
```

Without any arguments in MDEF, the default is the last referenced macro, namely "UNCLE".

TYPE IN MACRO **BODY.(<ALT>** *WHEN DONE):*

GODAD; ELDER;

Change ELDER to YOUNGER and type <ALT>

GODAD; YOUNGER; <ALT>

-DEFINED.

1:

Let us save this macro in a file called VISE.M

MSAVE("UNCLE");<CR>

OUTPUT FILE(NULL TO FORGET IT)=

VISE.M<CR>

SAVING UNCLE TO VISE.M

Note that MSAVE uses the current P-FILE if one is open.

1:

PSAVE;<CR>

This saves BASE-PLATE on VISE.M too;

1:

AL_WRITE;<CR>

To save the affixment structure in High level Al code.

OUTPUT FILE(NULL TO FORGET IT)=

VISE.AL3<CR>

1:

EXIT;<CR>

End of *SAIL* execution

End of session. The two files that are open will be automatically closed. On our next session DSKIN ("VISE.M") will define both UNCLE and BASE-PLATE.

POINTYCOMMAND SUMMARY

INTRODUCTORY INFORMATION ,

There are three arithmetic stacks, two of them being used for arithmetic operations, and the third for storing things that are popped off the first two in case at some future time you decide that' you didn't really mean to pop what you did. There are seven cursor stacks and the variable which contains the the pointer of the top element begins with the prefix CUR. We will mostly be concerned with the N stack and the D stack - the N stack acts as a general working register while the D stack is used together with the N in making affixments.

Any of the procedures after the DECLARATIONS section may be called by typing out the procedure name with the relevant parameters, and a semi-colon followed by a carriage return. The type of procedure is given as a matter of record for the advanced programmer.

The following sections of the command summary are arranged in the following order:

DESCRIPTION OF TERMS USED IN ARGUMENTS OF PROCEDURES
DECLARATIONS
NODE MANIPULATION
AFFIXMENT
MACRO DEFINITION AND MANIPULATION
ARITHMETIC
ARITHMETIC STACK OPERATIONS
CONNECTION OF AFFIXMENT STRUCTURE TO ARITHMETIC STACKS
ARM READING
ARM MOVEMENT
FILE INPUT OUTPUT
SPECIAL PARAMETERS
EXIT

DESCRIPTION OF TERMS USED IN ARGUMENTS OF PROCEDURES

( )

The term in the parentheses, usually called NULL below, contains the default argument.

STRING **STKID(NULL)**

This is the name-of the arithmetic stack used, and STKID ← "A:" or "B:". If no argument is given, the default stack is the last referenced stack. Initialization makes "A:" the default stack.

STRING **CID(NULL)**

This is the name of the cursor stack used, and CID← "N:","D:","P:","R:","M:","T:", or "K:". If no argument is given, the default stack is the last referenced stack. Don't forget the colon ":"

in both STKID and CID.

STRING NDSPC(NULL) or STRING NDSPC("N:")

Here the print name of the node or frame must be given in quotation marks, e.g., "VISE-JAW". If no argument is given, the default string is taken from the N stack, except in the case of procedure λ.

RPTR(NODE) VAL

This could be a procedure which generates a pointer to a node, or a pointer to a desired node. e.g. CTOP, λ("NODE") or CURNODE.

OPND

This is a type of data structure and members of the arithmetic stack are of this nature. XFELT, VECTOR, and SCALAR are all OPNDs.

Each member of the affixment structure is called a node, and its print name is the name we give the node. The elements of cursor stacks are nodes. The cursor top elements are also contained in the variables defined with the colon, e.g., "N:". The identifiers beginning with CUR are pointers to the top elements of cursor stacks. The actual names of the stacks in POINTY begin with $, e.g. the A stack is called $ASTACK, and the N stack is called $CURNODE, but when referenced by the user "N:" is considered to be the name of the stack.

Suppose we have a node called VISE-JAW which is the top element of the N stack, and its absolute location is on the top of the A stack. Then

The print name of the node is "VISE-JAW".
"N:" ≡ "VISE-JAW"
CURNODE ≡ λ("N:") ≡ λ("VISE_JAW")
ABSLOC("N:") • ATOP("A:")

where ≡ indicates that both sides of the equation have the same value. Thus, CPUSH(CURNODE), CPUSH(λ("N:") and CPUSH(λ("VISE_JA W")) will all have the same effect.

DECLARATIONS

This section gives a list of declarations made in POINTY and can be skipped for a first reading without much loss of understanding.

```
RCLASS NODE( STRING PNAME;RANY DAD,SON,EBRO,YBRO;
             INTEGER HOWLINKED; REAL ARRAY XF);
```

This is the definition of a record called NODE, showing the fields associated in the record, including the name of the node, information it has on any ancestor or son or elder or

younger brother, how it is linked, and an array giving the transformation.

These are the cursor stack declarations:

```
DCLSTK( CURNODE,NODE,4,"N:");      general working register;
DCLSTK( CURDAD ,NODE,4,"D:");      where subparts are to be affixed;
DCLSTK(CURPATH,NODE,4,"P:");       current name recognition subtree;
DCLSTK(CURREF  ,NODE,4,"R:");      current reference frame for motion;
DCLSTK( CURMOVE,NODE,4,"M:");      current motion frame;
DCLSTK( CURTREE,NODE,4,"T:");      current base node for display of tree;
DCLSTK( CURKILL,NODE,4,"K:");      magical kill stack;
```

These are the arithmetic stack declarations:

```
DCLSTK( ASTACK,OPND, 100,"A:");    -operand stack;
DCLSTK( BSTACK,OPND, 100,"B:");    operand stack;
DCLSTK( OSTACK,OPND, 100,"O:");    "oops" stack;
```

These are the stack indicator declarations:

```
RPTR( STACK) LASTCURSOR;           last cursor operated on;
RPTR( STACK) LASTARITH;            last arithmetic stack operated on;
RPTR( STACK) LASTSTACK;            last stack operated on;
```

These are the definitions of types:

```
DEFINE CURSORS "[]"
        =[ $CURNODE,$CURDAD,$CURPATH,$CURREF,
        SCURMOVE,SCURKILL,SCURTREE];
DEFINE OPND "[]" = [XFELT,VECTOR,SCALAR];
DEFINE ARITHS "[ 3" =[  SASTACK,SBSTACK,SOSTACK];
```

NODE MANIPULATION COMMANDS

RPTR(NODE) PROCEDURE λ(STRING NDSPC(NULL));

Pointer of node name stored in NDSPC. If the name is a cursor name, returns top of that cursor stack. A null argument will give the same pointer given by the previous call of λ. Note that the last node returned by λ appears on the display.

RPTR(NODE) PROCEDURE CITH(INTEGER I(0);STRING CID(NULL));

Returns the pointer to the ith element on the appropriate cursor stack. This instruction is useful when the element of interest is not on the top of the stack, and you do not want to upset the stack. Thus CITH(2,"N:") refers to the element labelled 2: in the N stack.

RPTR(NODE) PROCEDURE CPUSH(RPTR(NODE) VAL;STRING CID(NULL));
Pushes the pointer pointing to the desired node VAL into the appropriate cursor stack.

RPTR(NODE) PROCEDURE CPOP(STRING CID(NULL));
Pops the appropriate cursor stack.

RPTR(NODE) PROCEDURE CTOP(STRING CID(NULL));
Gets the top element of the appropriate cursor stack.

RPTR(NODE) PROCEDURE CROLLUP(STRING CID(NULL));
Rolls up all the elements of the appropriate cursor stack cyclically so that the top element goes to the bottom and the rest of the elements are pushed up one place.

RPTR(NODE) PROCEDURE CROLLDOWN(STRING CID(NULL));
Rolls down all the elements of the appropriate cursor stack cyclically so that the bottom element goes to the top and the rest of the elements are pushed down one place.

PROCEDURE CEXCH(STRING CID(NULL));
Exchange the two top elements of the appropriate cursor stack.

AFFIXMENT COMMANDS

PROCEDURE MK_NODE(STRING ID);
Defines a new node whose name is given by ID.

PROCEDURE COPY_NODE(STRING NDSPC("N:"));
Produces another node (a copy) on the N stack with the name given in NDSPC.

PROCEDURE NAME_NODE(STRING ID);
Renames' top node of N stack to the name specified in NDSPC.

PROCEDURE KILL(STRING NDSPC("N:"));
Kills the node named by NDSPC.

PROCEDURE UNKILL;
Retrieves the last killed node. Actually, it retrieves the node on the top of the K cursor stack.

PROCEDURE RIGID;
Attaches the node pointed to by N as a son of node pointed to by D rigidly. Represented by * sign.

PROCEDURE NONRIGID;
Attaches the node pointed to by N: as a son of node pointed to by D: non-rigidly. Represented by + sign.

PROCEDURE INDEPENDENT;
Attaches the node pointed to by N: as a son of node pointed to by D: independently. Represented by - sign.

PROCEDURE MERGE;
Merges the nodes pointed to by N: and D: cursors.

PROCEDURE GOSON(STRING CID(NULL));
The cursor goes to the son of the present node pointed to.

PROCEDURE GODAD(STRING CID(NULL));
The cursor goes to the dad of the present node pointed to.

PROCEDURE ELDER(STRING CID(NULL));
Goes to the node just below (not above) the present one if the next node is at the same level (i.e. is an elder brother), otherwise goes to a dummy node and the cursor will point to <empty>.

PROCEDURE YOUNGER(STRING CID(NULL));
Goes to the node just above (not below) the present one if there is one at the same level, otherwise goes to a dummy node and the cursor will point to <empty>.

MACRO DEFINITION AND MANIPULATION COMMANDS

The macro facility available is a primitive one and requires that parameters be stored on a macro parameter list.

INTEGER MPTOP; STRING ARRAY MPS[0:100];
MPTOP contains the position of the last parameter (last element) pushed into the macro parameter list, which is one less than the total number of parameters used since the parameter list begins at MPS(0). The maximum stack size of the macro parameter list is arbitrarily set to 100 at present.

PROCEDURE MDEF(STRING ID(NULL));
PROCEDURE MDEFQ(STRING ID, BODY);
Define a macro whose name is referred to as ID if there isn't one with that name present, or redefines the macro if it already exists - the default (in case no argument is included) is the last macro referertnced. The first allows the macro text to be typed in instruction by

instruction,the second enables the whole macro to be typed in and defined in the same line. There are some useful macro defintions on MACROS[PNT,RHT].

PROCEDURE MCALL(STRING ID(NULL));
Expands and executes the macro referred to as ID.

PROCEDURE MPUSH(STRING PARAM);
, Pushes the string PARAM on the macro parameter stack.

STRING PROCEDURES MP0,MP1,MP2,MP3;
Special procedures to return the top, 2nd, 3rd and 4th elements on the macro parameter stack.

STRING PROCEDURE MPGET(INTEGER I);
Get the Ith parameter on the macro parameter list, which is actually stored in MPS[MPTOP - I]. If I = 0 means the top of the parameter stack.

STRING PROCEDURE PROMPT(STRING S);
This procedure will output string S as a message and will return INCHWL (i.e. wait for a string to be typed at the terminal followed by a <CR>).

RECURSIVE PROCEDURE BCALL(STRING S 1 (NULL),S2(NULL));
This procedure will output string SI as a message and will accept a line of text S2 which it interprets and executes. If S2 finishes with !!GO; execution will resume where it left off, otherwise BAIL will prompt for more input. In that case, type EGO or <ALT>G to proceed.

Suppose we want to define a macro to construct the frame of a point but giving the user helpful advice in the process. One way of doing it is as follows:

```
MDEF( "CONSTRUCT-FRAME");
    BCALL( "POINT AT ORIGIN"); POINTIT;
    BCALL("POINT AT Z AXIS"); POINTIT;
    BCALL("POINT AT X-Z PLANE"); POINTIT;
    CONSTRUCT;
    MK_NODE(PROMPT("NODE NAME = "));
    ABSSET;
    CPUSH(λ( PROMPT( "DAD = "),"D:");
    BCALL( NULL,PROMPT( "AFFIXMENT = ")&";!!GO");
    APOP;
    <ALT>
```

An MCALL("CONSTRUCT_FRAME"); will wait for three prompts which must be replied to by !?GO or <ALT>G after telling where to point the pointer, and push the transforms into the arithmetic stack and then generate one transformation from these frames. Then there will be prompts for the node name, where it is to be affixed to and how it is to be affixed, and then then the frame is popped from the arithmetic stack. The <ALT> is

prompted for by the computer to end the macro definition.

The following example in which the nodes to which two cursors point have their pointers changed and illustrate the use of macro parameters.

```
MDEF( "EXCHANGE-POINTERS");
    CPUSH(λ(MP0),MP1);
    CROLLUP;
    CPOP(MP0);
    CPUSH(λ(MP1),MP0);
    CPOP(MP1);
    CROLLDOWN;
    MPTOP←MPTOP-2;
    <ALT>
```

A sample calling sequence to this macro would be as-follows:

```
MPUSH("N:");
MPUSH("D:");
MCALL("EXCHANGE_POINTERS");
```

This macro call will have the effect of changing the elements pointed to by D and N with each other. At the end of the execution, the macro parameter list will be popped.

ARITHMETIC COMMANDS

RPTR(XFELT) TR(REAL W,PH,TH,X,Y,Z);

Defines a trans which may be used as the first argument of APUSH. Note that XFELT is an OPND.

RPTR(VECTOR) PROCEDURE VE(REAL X,Y,Z);

Defines a vector with components x,y,z

RPTR(SCALAR) PROCEDURE SC(REAL VAL);

Defines a new scalar and pushes it on the top of the current arithmetic stack.

REAL PROCEDURE VMAGN(RPTR(VECTOR) V);

Returns the magnitude of vector V).

RPTR(VECTOR) PROCEDURE VADD(RPTR(VECTOR) V1,V2);

Returns a new vector which is the sum of the vectors V 1+V2.

RPTR(VECTOR) PROCEDURE VSUB(RPTR(VECTOR) V 1 ,V2);

Returns a new vector which is the difference of the vectors V I-V2.

RPTR(VECTOR) PROCEDURE NORM(RPTR(VECTOR) V);
Returns a new vector whose components are those of V but rormalited so that the magnitude is 1.

RPTR(VECTOR) PROCEDURE VCROSS(RPTR(VECTOR) V1,V2);
Returns a new vector which is the cross-product V 1 x V2.

REAL PROCEDURE VDOT(RPTR(VECTOR) V 1 ,V2);;
Returns the scalar dot product of vectors V 1 and V2.

RPTR(XFELT) PROCEDURE VVVTRANS(RPTR(VECTOR) A,B,C);
This creates a trans with origin at A, z-axis through B, x-t plane through C. CONSTRUCT makes use of this procedure by making using of the x,y and z coordinates of the three transes on the top of the appropriate arithmetic stack, popping them, and pushing the result on the top of the stack.

Note that no special commands for arithmetic operations on scalars have been defined, since BAIL is able to do routine arithmetic computations. To find the value of an arithmetic expression, simply type the expression followed by a ";" and a carriage return, and the value will be given.

ARITHMETIC STACK OPERATIONS

RPTR(OPND) PROCEDURE AITH(INTEGER I(0);STRING STKID(NULL));
This is a reference to elements in the appropriate arithmetic stack when the element is not at the top of the stack. AITH(2,"A:") refers to the element labelled 2: in the A stack. Note that the argument refers to the current position in the stack, and that this prodecure does not alter the stack in any way. An example on its use as as follows:

APUSH(AITH(2,"B:"),"A:");
This has the effect of pushing the. element labelled 2: in the B stack onto the top of the A stack.

RPTR(OPND) PROCEDURE APUSH(RPTR(OPND) VAL;STRING STKID(NULL));
This pushes the transform given by VAL on the arithmetic stack. Be careful with the first argument: it is the pointer to an OPND and should be a procedure that generates such *a* pointer, e.g. ATOP or TR( ).

RPTR(OPND) PROCEDURE APOP(STRING STKID(NULL));
Pops the top element of the appropriate arthmetic stack.

RPTR(OPND) PROCEDURE AFLUSH(STRING STKID(NULL));
Like APOP except doesn't save anything on the 0 stack.

RPTR(OPND) PROCEDURE **ATOP(** STRING **STKID(** NULL));
Gets the pointer to the top element of arithmetic stack.

RPTR(OPND) PROCEDURE **AROLLUP(** STRING **STKID(** NULL));
Rolls up all the elements of the appropriate arithmetic stack cyclically so that the top element **becomes** the bottom element, and the other elements are all shifted one element.

RPTR(OPND) PROCEDURE **AROLLDOWN(** STRING **STKID(** NULL));
Rolls down all the elements of the appropriate arithmetic **stack** cyclically so that the bottom element becomes the top element, and the other elements are all shifted down one element.

PROCEDURE AEXCH(STRING **STKID(NULL));**
Exchange the top two elements of the appropriate arithmetic stack.

PROCEDURE **TMUL(** STRING **STKID(NULL));**
Multiply the top two elements of the appropriate stack and pop them, and push the answer into the stack.

PROCEDURE **TINV(** STRING **STKID(NULL));**
Replace the top element of the appropriate stack with the inverse transform.

PROCEDURE **TEDIT(** STRING **STKID(** NULL));
Puts the top element of the appropriate stack into the line editor with the instruction to push it back onto the stack after editing or correcting.

PROCEDURE **OOPS(** STRING **STKID(** NULL));
Gets back the **value** of the element we just popped from the appropriate stack.

PROCEDURE CONSTRUCT(STRING **STKID(NULL));**
This constructs an implicit frame from the top three frames on the last arithmetic stack referenced. The three frames are popped off, and the new implicit frame is pushed on.

PROCEDURE **VA(** STRING **STKID(** NULL));
PROCEDURE **VS(** STRING **STKID(** NULL));
PROCEDURE **VM(** STRING **STKID(NULL));**
PROCEDURE **VC(** STRING **STKID(NULL));**
PROCEDURE **NV(** STRING **STKID(** NULL));
PROCEDURE **VD(STRING STKID(NULL));**
These procedures have the same functions **as** VADD, VSUB, VMAGN, VCROSS, NORM, and VDOT respectively, except that the operands are popped off the relevant arithmetic stack, and the result then pushed into the stack. In the case where two operands are necessary, **V2** corresponds to the top element of the stack, while vl corresponds to the next **element.**

PROCEDURE PV(STRING STKID(NULL));
If the top element of the appropriate arithmetic stack is a TR, then it is popped off, and the position coordinates are left on the top of the stack. If the top element is not a trans, an error message is returned.

COMMANDS TO CONNECT AFFIXMENT STRUCTURE TO ARITHMETIC STACKS

RPTR(XFELT) PROCEDURE ABSLOC(STRING NDSPC("N:"));
Absolute location of a node, the default node being the node pointed to by the "N:" cursor. The argument must be the name of a node that has been previously defined.

RPTR(XFELT) PROCEDURE RELLOC(STRING NDSPC("N:"));
Similar to ABSLOC except the relative transform of the node with respect to its parent is returned.

PROCEDURE ABSSET(STRING NDSPC("N:"),STKID(NULL));
Sets absolute location of the appropriate node (default is where N points) as the value on the top of the appropriate arithmetic stack.

PROCEDURE RELSET(STRING NDSPC("N:"),STKID(NULL));
Sets relative location of the appropriate node (default is where N points) as the value on the top of the appropriate arithmetic stack.

ARM READING COMMANDS

PROCEDURE READARM;
Reads the current position of the arm.

PROCEDURE ATFID;
Asserts that the pointer is at fiducial and updates the value of the arm.

PROCEDURE DEFFID;
Very first step; define fiducial with respect to world. This procedure asserts that the fiducial is currently at the ARM frame.

PROCEDURE POINTIT(STRING STKID(NULL));
Reads position at the end of the pointer and pushes it into the appropriate arithmetic stack. STKID is either "A? or "B:";lower case a or b invalid.

PROCEDURE **GRABBIT(** STRING **STKID(NULL));**
Reads position at the finger and pushes it into the appropriate arithmetic stack.

PROCEDURE **HERE(** STRING NAME);
Defines a new node called NAME and puts the current position of ARM into it.

ARM MOVEMENT COMMANDS

(These do not work on the Blue Arm at present.)

. PROCEDURE **GOARM(REAL** ARRAY BXF);
This moves the arm to the 4x4 transformation given by BXF.

PROCEDURE **MOVEABS(** STRING **STKID(** NULL));
This moves the frame pointed to by **CURMOVE** to the frame specified in the arithmetic stack. With no stack defined the appropriate stack is the last referenced arithmetic stack.

PROCEDURE **MOVEREF(** STRING **STKID(** NULL));
This moves the frame pointed to by **CURMOVE** to the frame specified in the arithmetic stack assuming that the **latter** frame is with respect to a co-ordinate system pointed to by CURREF.

PROCEDURE **MOVEREL(** STRING **STKID(** NULL));.
This moves the frame pointed to by **CURMOVE** by a amount specified on the Arithmetic Stack assuming that that value **is on** a co-ordinate system pointed to by CURREF.

PROCEDURE FREE;
This frees the arm for 5 seconds, during which time the user should move the arm to a desired location and push the panic button. The absolute frame of the arm is then updated. If instead there is a time-out without the panic button being pushed, nothing happens.

PROCEDURE **DMOVE(REAL X,Y,Z);**
Move the frame pointed to by **CURMOVE** differentially by **x,y,z** in the **x,y,z** directions respectively.

PROCEDURE **DX(REAL X);**
Move the frame pointed to by **CURMOVE** differentially in the x direction by quantity specified.

PROCEDURE **DY(REAL Y);**
Move the frame pointed to by **CURMOVE** differentially in the y direction by quantity specified.

PROCEDURE DZ(REAL Z);
Move the frame pointed to by CURMOVE differentially in the z direction by quantity specified.

FILE INPUT/OUTPUT COMMANDS

PROCEDURE AL-WRITE;
Dumps into an AL-file in high-level' AI code the affixment structure pointed to by CURNODE.

PROCEDURE AL-CLOSE;
Closes the file containing the AL declarations of the data structures.

PROCEDURE PSAVE(STRING NDSPC("N:"));
Dumps out all the POINTY instructions necessary to generate the code needed to obtain the affixment associated with the node NDSPC in case we lose everything carelessly into a P_file - if there is no P_file open PSAVE will take the necessary steps to open one; the default node is the node pointed to by CURNODE.

RECURSIVE PROCEDURE SAVE_NODE(RPTR(NODE) ND);
Dumps out into a P_file the affixment tree rooted at node ND. This routine is called by PSAVE, which it is more desirable to call. RPTR(NODE) ND can be either of the form CURNODE, CURDAD,etc or λ("N:") or λ("NODE").

PROCEDURE P-CLOSE;
Closes the currently open P_file.

RECURSIVE PROCEDURE DSKIN(STRING FID);
Reads in and executes POINTY instructions from a disk file FID to generate affixment structure(s) and set up the macros.

PROCEDURE MSAVE(STRING ID(NULL));
Save macro ID onto P_file. If one is not currently open, instructions. to open one will be given. "*" will dump all the macros. A null argument will dump the last macro referenced.

SPECIAL PARAMETERS

INTEGER UPDSUPPRESS,TISUPPRESS;
, If UPDSUPPRESS>0 then do not display anything. UPDSUPPRESS is incremented by integer TISUPPRESS at the start of a macro expansion or DSKIN and restored to the

previous value upon exit. Thus, setting TISUPPRESS←0; will allow you to observe successive steps in the macro expansion.

BOOLEAN SHOWXFS,SHOWLINKS;
These control the display (display if TRUE, suppress if FALSE) of the Transes and the Link structures of nodes (the last used for debugging purposes) respectively.

REAL π;
POINTY knows the value of π to be 3.14 1592653.

EXIT COMMAND

. PROCEDURE EXIT;
Exits from POINTY, and closes any output files that might be open.

ACKNOWLEDGEMENTS

The author would like to express his thanks to Russell Taylor for valuable suggestions and help with POINTY, and to Dave Grossman for editorial help.

IX. MONTE CARLO SIMULATION OF TOLERANCINC

David D. Grossman

Artificial Intelligence Laboratory
Computer Science Department
Stanford University

The author is a Research Staff Member in the Computer Science Department, **IBM T. J. Watson** Research Center, P. 0. Box **218,** Yorktown Heights, N. **Y.** 10598. At **the time this** research was performed, he was a visiting Research Associate in the **Computer Science** Department at Stanford University on sabbatical from IBM.

INTRODUCTION

The assembly of discrete parts is a major fraction of industrial production. The role of computers in this field has been limited primarily to production and inventory control, computer aided design, and programming numerically controlled machine tools. Very little progress has been made in applying computers to the problem of simulating assembly processes, in spite of the fact that such simulation offers the possibility of considerable savings over the alternative cost of building pilot production tines.

When one examines other large industrial fields one finds that computer simulation is a much more widely used tool. There are basically two reasons, however, why this tool has not been extensively applied in discrete parts assembly. First, because assembly is not a scientific discipline, experience is formulated as a set of *ad hoc* principles rather than as a mathematical theory. Although such principles may be set forth in textbooks,[1] it is difficult to embody them in computer simulations. This situation is in sharp contrast, for example, to the way differential equations can be used to model complex chemical processes. The second reason is that assembly environments Contain an immense variety of dissimilar objects. This aspect of assembly is in sharp contrast, for instance, to nuclear physics simulations where all neutrons behave in the same way.

The only obvious unifying principle in discrete parts assembly is that in 3-dimensional space no two objects may intersect. This fact suggests a formulation of the simulation problem in terms of set theory, an approach which is being taken in research on parts description at the University of Rochester.[2,3,4] Set theoretic representations are good for determining if a given point is inside a particular set, but performance difficulties arise on problems involving pairs of sets. For example, the question of whether or not a piston intersects a motor block is difficult to answer because it is likely to cause a lengthy search for a point contained in both sets. Compounding this difficulty is the fact that assembly involves continuous motion of the discrete parts, so that it is desirable to be able to solve set intersection problems at every instant of time. The computational algorithms would not be hard to formulate, but the execution times would be extremely long, even on the fastest computers in the world. For this reason, simulation of the full assembly process is intractable, although simulation of special classes of assembly problems is still a practical and achievable goal.

From among the many aspects of assembly which could conceivably be modeled, this paper is concerned with the implications of tolerancing and imperfection. In the literature on this subject, dimensional tolerancing has Come to mean specifying the tolerances of parts in mechanical drawings. A national standard has been established which defines the meanings of tolerancing symbols in drawings[5] and textbooks have been written to explain the use of these symbols.[6] The emphasis on drawings, however, tends to obscure the underlying reasons for being concerned with tolerances. The issue is not so much what 3.000±.005 cm means but rather *why* the designer chose to specify this tolerance in the first place.

There are three factors which enter into specifying tolerances in drawings. First, the discrete part which is described must ultimately be assembled into a product which is expected to have some function, and the tolerance may be needed to provide this function. For instance, it is highly desirable that each chamber of a Colt revolver align accurately with the barrel. Secondly, the part may be required to have certain tolerances in order that the assembly process itself be feasible. For example, in order to assemble an automobile engine, the holes in the gasket must align with those in the block. Also, it is often necessary to have very accurate parts to avoid jamming vibratory feeders. In fact, it is often necessary to design higher precision into the assembly process than is functionally needed in the final product. Finally, tolerances may be assigned to correspond to the capabilities of the manufacturing method chosen. Tolerances achievable by sheet metal stamping would not be the same as those achievable on a numerically controlled machine tool, and it would be foolish to assign tolerances in a drawing which would give unreasonably small yields.

The' product designer uses his expertise in product design, assembly, and manufacturing to specify tolerances in the drawing which are both-adequate and achievable. An excellent textbook has been published which describes the considerations involved in this process.[7] The process is complicated because the design criteria depend *on* the combined tolerances, rather than on the tolerances individually. Typically, the designer must trade off between alternative ways of selecting individual tolerances in order to achieve some resultant tolerance with minimum cost. Unfortunately, the S-dimensional relationships involved are usually too tedious to allow a rigorous mathematical treatment in all but the simplest cases. The designer therefore uses a great deal of intuition in reaching a decision. Finally he writes down a number like 3.000f.005 cm and throws away all the information which went into this decision.

A recent paper from General Motors describes a system which enables product designers to specify a set of individual parts tolerances and simulate the stochastic properties of interesting resultant tolerances.[8] The system is based on the Monte Carlo method, a simulation technique which is well known and has been widely used in many other applications.[9] The existence of the GM paper shows that a need exists for simulation tools in the field of parts tolerancing. The problem of tolerancing is sufficiently hard, and the stakes are sufficiently high, that intuition is no longer a satisfactory method for specifying parts tolerances.

The approach taken in the GM work is to provide an interactive system in which the user can obtain high statistics very quickly. In order to achieve execution speed, the user must explicitly provide all the equations which tell how the resultant tolerances depend on the individual tolerances. The system models just the positions and orientations of a few features of the part, rather than the entire part shape. This system is apparently proving quite useful to GM designers.

Aside from the GM work, the only other published papers relating to modeling parts tolerances are those from the University of Rochester, *where a* language called PADL for

representing a class of discrete parts is being developed.[2,3,4] The hope is that PADL descriptions can someday be used to generate programs for numerically controlled machine tools which can make the parts.

The topic of parts representation without regard to tolerancing has been studied by Binford, Agin, and Nevatia,[10,11,12] Braid,[13,14] Baumgart,[15,16] Grossman,[17] and Lieberman and Lavin.[18,19] Although none of these parts modeling schemes was designed with tolerancing in mind, both the Baumgart and Grossman approaches offer a natural way of adding Monte Carlo procedures to simulate tolerances. As the author of one of these papers, my choice of which of the two systems to use for the current work was highly biased. I chose to use my own system solely because I am much more familiar with it.

Although the balance of this paper describes a specific implementation of Monte Carlo tolerancing within a parts representation system, many of the issues discussed are implementation independent. The point of this paper is not simply to give a blueprint for a specific way of simulating tolerances but rather to show that such a system is possible, to expose some of the design issues, and to give examples of ways in which the system might be used.

The simulation method described in this paper most closely resembles that of the GM paper, but there are several major differences. Whereas the CM system computes the resultant tolerances of individual parts from tolerances specified in mechanical drawings, the current work is much more comprehensive. It allows one to simulate the propagation of tolerances all the way from the manufacturing process right through the assembly process. Also, while the GM work requires that the user explicitly supply formulas for the resultant tolerances as functions of the individual tolerances, the current work provides system routines which automatically perform these sorts of operations numerically. This provision is particularly useful because in many situations the relevant formulas can not be derived in closed form. On the other hand the GM system is interactive, runs at high speed, and yields high statistics answers, while the current system runs in batch mode, executes much more slowly, and therefore yields much poorer statistics.

The next section of this paper reviews the- main features of my earlier publication on representing parts by PL/I procedures and explains how this system can easily be applied to the Monte Carlo simulation of parts tolerances. This method is then illustrated by four specific examples, one of which is chosen from the field of assembly by computer controlled manipulators. The reason for chosing this example is that this research was carried out as part-of continuing manipulator projects at the IBM T. J. Watson Research Center and the Stanford University Artificial Intelligence Laboratory. However, it is important to stress that the simulation techniques described here are applicable not only in the domain of computer controlled assembly, but also in the much wider domain of manufacturing and assembly as they exist in industry today, using conventional equipment and procedures. The paper closes with a discussion of research areas appropriate for extension of the Monte Carlo tolerancing method.

MONTE CARLO METHOD

Distributions

The basic idea of any Monte Carlo calculation is to generate an ensemble of models which simulates an ensemble of real entities.[9] The statistical properties of the real entities may then be simulated by studying the corresponding properties of the models. Such simulation is useful when purely analytical methods cannot be found.

For the case of discrete parts manufacturing and assembly, the real entities consist of three-dimensional objects at a workstation. These objects include component parts and their features, tools and fixtures, measuring instruments, and automation equipment up to the level of complexity of transfer lines and computer controlled manipulators. For all of these objects, the primary attributes to be modeled are shape, position, and orientation.

In simulating statistical distributions of shape, position, and orientation attributes, it is necessary to define the meaning of expressions Of the form 3.000±.005 cm. One possible definition would be a normal distribution with a mean of 3.000 cm and a standard deviation of which .005 cm is some small integral multiple. This choice would allow dimensions to fall outside the specified range, albeit infrequently. Another possibility would be to have a distribution which goes rigorously to zero outside the specified range. Inside the range, the distribution could be uniform, or peaked at 3.000 cm, or bimodally peaked at 2.995 cm and 3.005 cm. The distribution function might also be skewed if, for example, a part has been manufactured in a fixture which is showing signs of progressive wear.

The ANSI dimensioning and tolerancing standards do not specify what statistical distribution is implied by expressions of the form 3.000±.005 cm.[5] This omission is actually necessary, because the shape of the distribution function depends on the manufacturing process, so that the choice of this shape is best left to the production engineer. In the system described in this paper, an arbitrary choice was made to restrict the class of allowed distributions to be either uniform or normal. This choice was made for the sake of convenience and does not represent any inherent limitation in the method.

Part Ensembles

In most parts modeling systems the user describes each part in terms of numbers which are entered directly into a data structure. This data structure, therefore, represents a particular *instance* of a part rather than an *ensemble* of similar parts. For the Monte Carlo simulation of tolerances, however, it is necessary that the parts modelihg system provide some simple means of representing ensembles. What is needed, therefore, is a system in which the user describes parts not in terms Of *numbers* but in terms Of *parameters* that are assigned numerical values when a part is instantiated. The advantage of such a system for this Monte Carlo simulation is that a random number generator may be used to assign values to these

parameters.

The use of parameters to characterize arbitrary attributes of parts is one of the principle features of the Procedural Geometric Modeling System (PGMS) developed earlier by this author! [17] This modeling system was therefore used for the current study, The reader is referred to the earlier publication for details concerning the way in which PGMS represents 3-dimensional objects as PL/I procedures. A brief summary of the main features of this system are included here for the sake of completeness. Further features will be explained in subsequent sections of this paper as the need arises.

In PGMS, a hypothetical part whose name is "widget" and which has two attributes might be invoked by the calling sequence

```
CALL SOLID(WIDGET,A,B);
```

The generic widget itself would be represented by a PL/I procedure whose entry point is named WIDGET. This procedure would describe how the widget is hierarchically constructed out of its component subparts. These subparts might be positive SOLID's or negative HOLE's. For example,

```
WIDGET: ENTRY (A,B);
CALL SOLID(CUBOID,A,A,B);
CALL HOLE(CUBOID,A,A/2,B-10);
RETURN;
```

A library of parts procedures already exists which starts with the primitive POINT and includes such objects as LINE, CUBOID, CONE, WEDGE, CYLNDR, and HEMISPH. More complicated objects have also been coded, up to the level of complexity of IMM, which represents the IBM Research mechanical manipulator, and SUARM, which represents the Stanford University arm.

In addition to parts procedures, PGMS provides routines to perform transformations in 3-dimensional space. For example, if the generic widget were translated by C units along the Y-axis and then rotated by D degrees about the X-axis, the calling sequence would be

```
CALL YTRAN(C);
CALL XROT(D);
CALL SOLID(WIDGET,A,B);
```

A particular instance of a widget would be invoked by assigning values to the parameters. For example,

```
CALL YTRAN( 12);
CALL XROT(30);
CALL SOLID(WIDGET,3.000,16.5);
```

An ensemble of 500 similar widgets would be represented by the calling sequence

```
DO I=1 TO 500;
    CALL YTRAN(12*RAND(-0.1,+0.3));
    CA LL XROT(30+GA USS(2.5));
    CALL SOLID(WIDGET,3.000+RAND(-.005,+.005),16.5+RA ND(-.2,+.2));
    END;
```

where the function RAND(X,Y) returns a random number uniformly distributed on the interval from X to Y, and the function GAUSS(Z) returns a random number normally distributed with mean 0 and standard deviation Z.

Semantics

Once an ensemble of parts has been represented, PGMS provides a way to derive properties from the representation. This process is referred to as attaching semantics to the representation. The first step is to code a semantic routine which can compute a desired property. For example, the routine TOTVOL shown below adds up the volume Of all positive and negative CUBOID's in any object.

```
TOTVOL: PROCEDURE (NODE,X,Y,Z);
DECLARE NODE ENTRY;
IF NODE=CUBOID THEN VOLUME=VOLUME+POLARITY*X*Y*Z;
RETURN;
END TOTVOL;
```

Next, calls to system routines BEGIN, EXEC, and END are used to attach these semantics to the system and the part procedure of interest is executed. In the case of the ensemble of 500 widgets, one could print the volume of each widget with the following code.

```
DO I- 1 TO 500;
    VOLUME-O;                    /*INITIALIZE VOLUME*/
    CALL BEGIN(5000);            /@ALLOCATE STORAGE+/
    CALL EXEC(TOTVOL);           /*ATTACH SEMANTICS*/
    CALL YTRAN(12+RAND(-0.1,+0.3));
    CA LL XROT(30+GA USS(2.5));
    CALL SOLID(WIDGET,3.000+RAND(-.005,+.005),16.5+RAND(-.2,+.2));
    CALL END;                    /*DEALLOCATE STORAGE*/
    PUT SKIP DATA (VOLUME);      /*PRINT WIDGET VOLUME*/
    END;
```

Generalizing from this example, one can easily see how to provide semantics to display histograms of almost any desired properties of the ensemble. What is probably not clear from this example is the fact that for more realistic parts, the hierarchy of subpart calls

involves so much computation that execution is usually rather slow. For instance, when the procedure for the Stanford arm is executed on an IBM 370/168 running the VM time sharing system with 120 users, each instantiation takes about 6 seconds of virtual CPU time and 1 minute of elapsed time. Deriving the properties of an ensemble of 500 Stanford arms would therefore require about 8 hours of elapsed time. This number is prohibitively long for casual use of the system. However, 8 hours of elapsed time in simulating a complex mechanism would certainly not be excessive if the derived properties were to reveal a design deficiency which would have taken months to correct had the hardware been built first.

Another fact which is not clear from the example above is that parts of typical complexity require the allocation of several hundred thousand bytes of intermediate storage. The Stanford arm procedure, for instance, requires nearly 300K of storage. The reason behind this need for intermediate storage relates to the detailed implementation of PGMS, a topic which is discussed in my prior publication and which will be omitted here.

EXAMPLES

Rivet-Hole Bracket

The first example chosen to illustrate Monte Carlo tolerancing in PGMS is similar to the rivet-hole bracket used as the example in the GM paper. A few changes were made because the original drawing shows only a partial view of the bracket in two dimensions, while in PGMS it is desirable to model the part completely and in three dimensions.

The modified rivet-hole bracket may be represented by the following code:

```
RHBRAK: ENTRY (X1,Y1,RAD1,X2,Y2,RAD2,ANG,THICK,LENG,NSECT);
DECLARE RHBFRAME(4,4) FLOAT;
CALL STORE(RHBFRAME);
CALL SOLID(WEDGE,THICK,LENG,ANG,1);        /*BRACKET*/
CALL XYZTRAN(X1,Y1,0);
CALL HOLE(CYLNDR,THICK,RAD1,NSECT);        /*HOLE 1*/
CALL RECALL(RHBFRAME);
CALL XYZTRAN(X2,Y2,0);
CALL HOLE(CYLNDR,THICK,RAD2,NSECT);        /*HOLE 2*/
RETURN;
```

The call in the above code to the PGMS routine STORE is used to save the current coordinate frame in the local array RHBFRAME. Subsequently, the current frame is translated from the corner of the bracket to the position of the first hole. The current frame is then returned to the bracket corner by the RESTORE routine, so that it may subsequently be translated to the position of the second hole.

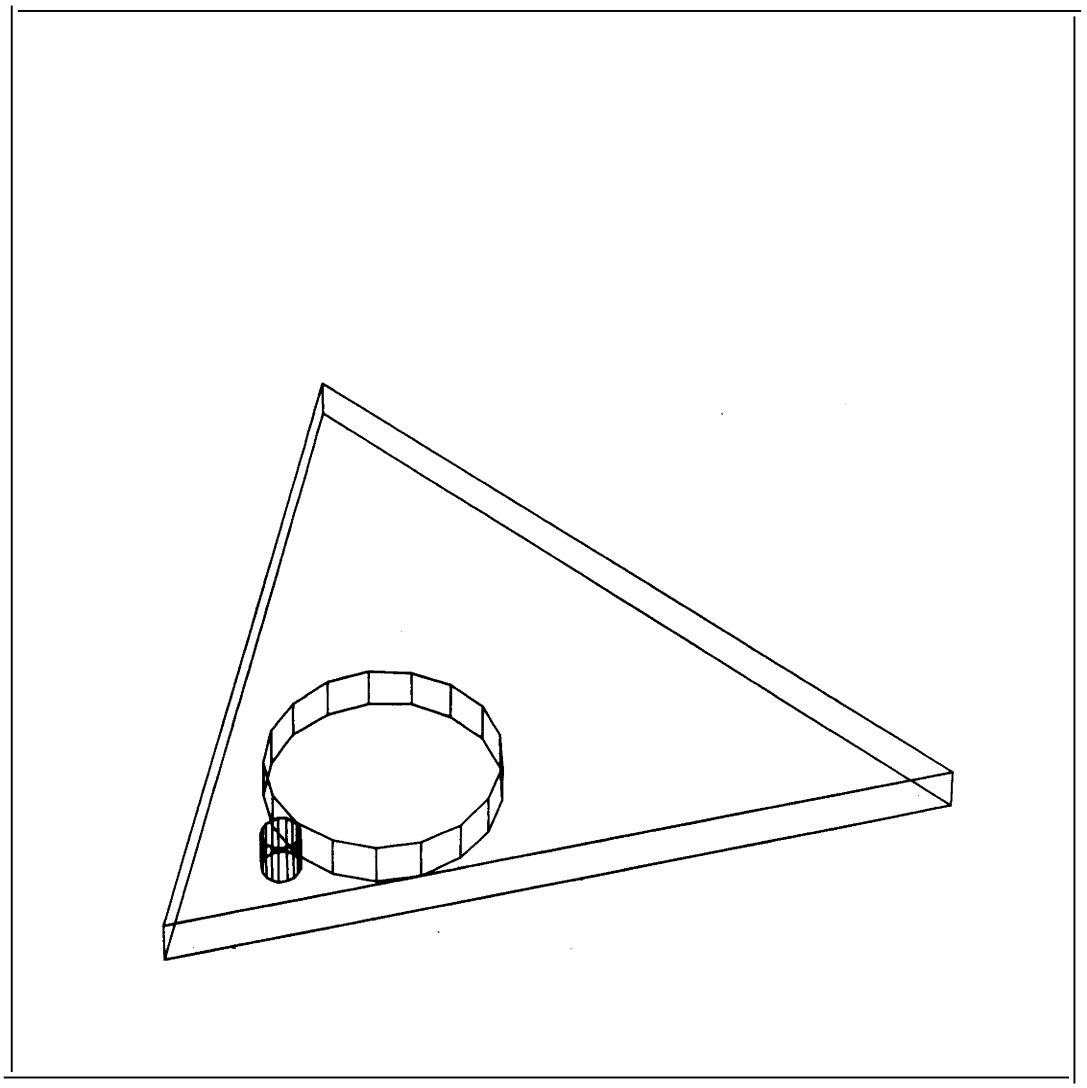


Figure 1: Drawing of **Rivet-Hole** Bracket

RHBRAK ( )
  WEDGE (1)
    GLINE (1,1)
      LINE (1,1,1)
        POINT (1,1,1,1)
        POINT (1,1,1,2)
    GLINE (1,2)
      LINE (1,2,1)
        POINT (1,2,1,1)
        POINT (1,2,1,2)
    GLINE (1,3)
      LINE (1,3,1)
        POINT (1,3,1,1)
        POINT (1,3,1,2)

    . . . (a total of 9 GLINE'S)

  CYLNDR (2)
    GLINE (2,1)
      LINE (2,1,1)
        POINT (2,1,1,1)
        POINT (2,1,1,2)

    . . . (a total of 3*NSECT GLINE'S)

  CYLNDR (3)
    GLINE (3,1)
      LINE (3,1,1)
        POINT (3,1,1,1)
        POINT (3,1,1,2)

    . . . (a total of 3*NSECT GLINE'S)

Figure 2: Rivet-Hole Bracket Subpart Hierarchy

The ten parameters of this procedure represent the seven dimensions subject to tolerancing, the part thickness and length, and the number of sectors used in approximating the cylindrical holes by polyhedra. The effect of this polyhedral approximation can be seen in Figure 1 which was generated by attaching a standard graphics semantic routine to the RHBRAK procedure.

The RHBRAK procedure represents a subpart hierarchy of 40+24*NSECT nodes as indicated in Figure 2. At the top level, the RHBRAK consists of a solid WEDGE and two CYLNDR holes. The WEDGE in turn is composed of nine GLINE's (general lines), each of which is made out of one LINE with two end POINT's. Every level in this hierarchy can be referred to by a unique *subaddress*, also shown in Figure 2. For instance, the LINE along the bottom left edge of the RHBRAK has a subaddress of (1,3,1). The importance of these subaddresses will become clearer in the discussion which follows.

In the GM paper, the designer is concerned with the clearance between the two holes and the clearances between the second hole and the edges of the part. In order to study these resultants, the following semantic routine might be used.

```
BRA KRES: PROCEDURE (NODE,X1,Y1,RAD1,X2,Y2,RAD2);
DECLARE (RGHTEDGE,LEFTEDGE,HOLE1,HOLE2) POINTER;
DECLARE NODE ENTRY;
IF NODE-RHBRAK THEN DO;
     CALL DEFINE (RGHTEDGE,1,2,1);
     CALL DEFINE (LEFTEDGE, 1,3,1);
     CALL DEFINE (HOLE 1,2);
     CALL DEFINE (HOLE2,3);
     CLEAR 1-DISTOO(HOLE1,HOLE2)-RAD1-RAD2;
     CLEAR2-DISTOX(HOLE2,RGHTEDGE);
     CLEAR3-DISTOX(HOLE2,LEFTEDGE);
     END;
RETURN;
END BRAKRES;
```

The DEFINE routine of PGMS is used to associate a PL/I pointer variable with any previously specified frame in the part hierarchy. The first argument in the call to DEFINE gives the name of the pointer variable and the subsequent arguments give the subaddress in the part hierarchy. Encoding these subaddresses requires that the user have a manual which summarizes the subpart hierarchy generated by each procedure in the part library and shows drawings of the basic volume shapes. Understanding subaddresses is currently the most tedious aspect of PGMS.

The function DISTOO invoked in this semantic routine returns the distance from Origin to Origin (00) of the two specified frames. The function DISTOX returns the distance from Origin to X-axis (OX) of the two specified frames. In order to have written this code it is necessary to have known that every LINE runs along the X-axis of its frame, and that every

CYLNDR runs along the positive Z-axis of its frame. Thus CLEAR 1, CLEAR2, and CLEAR3 are the desired clearances. It can be seen from this example that the polyhedral approximation has absolutely no effect on the statistical properties of these clearances.

Finally, a short program may be written to attach these semantics to the system and print the three clearances for each of 500 rivet-hole brackets.

```
DO I=1 TO 500;
    CALL BEGIN(50000);
    CALL EXEC(BRAKRES);
    CALL SOLID(RHBRAK,1.325+GAUSS(.005/3),        /*X1*/
                      .875+GAUSS(.005/3),         /*Y1*/
                      .2+RAND(-.0075,.0075),      /*RAD1*/
                      2.525+GAUSS(.005/3),        /*X2*/
                      1.615+GAUSS(.005/3),        /*Y2*/
                      1.2+RAND(-.0075,.0075),     /*RAD2*/
                      67+RAND(-.25,+.25),         /*ANG*/
                      0.25,                       /*THICK*/
                      8.0,                        /*LENG*/
                      1);                         /*NSECT*/
    CALL END;
    PUT SKIP DATA (CLEAR1,CLEAR2,CLEAR3);
    END;.
```

Because execution time varies roughly in proportion to the total number of nodes in the subpart hierarchy, NSECT has been set to 1 here. This simulation of 500 rivet-hole brackets takes about 3 minutes of CPU time on an IBM 370/168.

For this example, using PGMS to model tolerances is somewhat more difficult than using the GM system, largely because the tedium of understanding subaddresses outweighs that of writing down a few trigonometric formulas. As the examples become more complicated, however, the subaddress problem remains about constant, while the trigonometry problems become much worse. The overall balance therefore swings in favor of PGMS.

Box Manufacture

This example of Monte Carlo tolerancing is concerned with a manufacturing process in which 4 holes *are* drilled into a rectangular box. The holes are made by a gang drill with drill bits held in four separate chucks, while the box is held in a fixture attached to the drill bed. The box is 12 cm long, 8 cm wide, and 4 cm. high, and the four corner holes have radius 3 mm and depth 2.5 cm and are nominally 1 cm from each edge.

Tolerance errors in the positions of the holes are generated because the fixture may be translated or rotated slightly in the plane of the drill bed, and each of the four drill chucks

may be' radially displaced slightly from its nominal position. To make the example somewhat more interesting, it will be assumed that the rotational error in positioning the fixture is about an axis which runs through a corner rather than the center of the box.

Each of the drill bits may be modeled as a cylinder which has been radially displaced by RADERR in a random direction from its desired position.

```
DRILBIT: ENTRY (RADERR,LENG,RAD,NSECT);
CALL ZROT(RAND(0,360));
CALL XTRAN(RADERR);
CALL SOLID(CYLNDR,LENG,RAD,NSECT);
RETURN;
```

An ensemble of boxes manufactured by this process may then be represented as a cuboid with holes cut out by the four drill bits.

```
RECTBOX: ENTRY (X,Y,Z,LENG,RAD,NSECT,
                XERR,YERR,ANGERR,RADERR);
CALL SOLID(CUBOID,X,Y,Z);                          /*BLOCK*/
CALL ZROT(ANGERR);
CALL XYZTRAN(XERR,YERR,Z-LENG);
CALL XYZTRAN(1,1,0);
CALL HOLE(DRILBIT,RADERR,LENG,RAD,NSECT); /*HOLE 1*/
CALL XTRAN(X-2);
CALL HOLE(DRILBIT,RADERR,LENG,RAD,NSECT); /*HOLE 2*/
CALL YTRAN(Y-2);
CALL HOLE(DRILBIT,RADERR,LENG,RAD,NSECT); /*HOLE 3*/
CALL XTRAN(2-X);
CALL HOLE(DRILBIT,RADERR,LENG,RAD,NSECT); /*HOLE 4*/
RETURN;
```

The next step is to code a semantic routine which can derive the coordinates of the four holes with respect to the coordinate system of the box.

```
HOLFIND: PROCEDURE (NODE);
DECLARE (HOLE 1,HOLE2,HOLE3,HOLE4) POINTER;
DECLARE NODE ENTRY;
IF NODE-RECTBOX THEN DO;
     CALL DEFINE (HOLE 1,2); CALL ORIGIN (HOLE 1,POS 1);
     CALL DEFINE (HOLE2,3); CALL ORIGIN (HOLE2,POS2);
     CALL DEFINE (HOLE3,4); CALL ORIGIN (HOLE3,POS3);
     CALL DEFINE (HOLE4,5); CALL ORIGIN (HOLE4,POS4);
     END;
RETURN;
END HOLFIND;
```

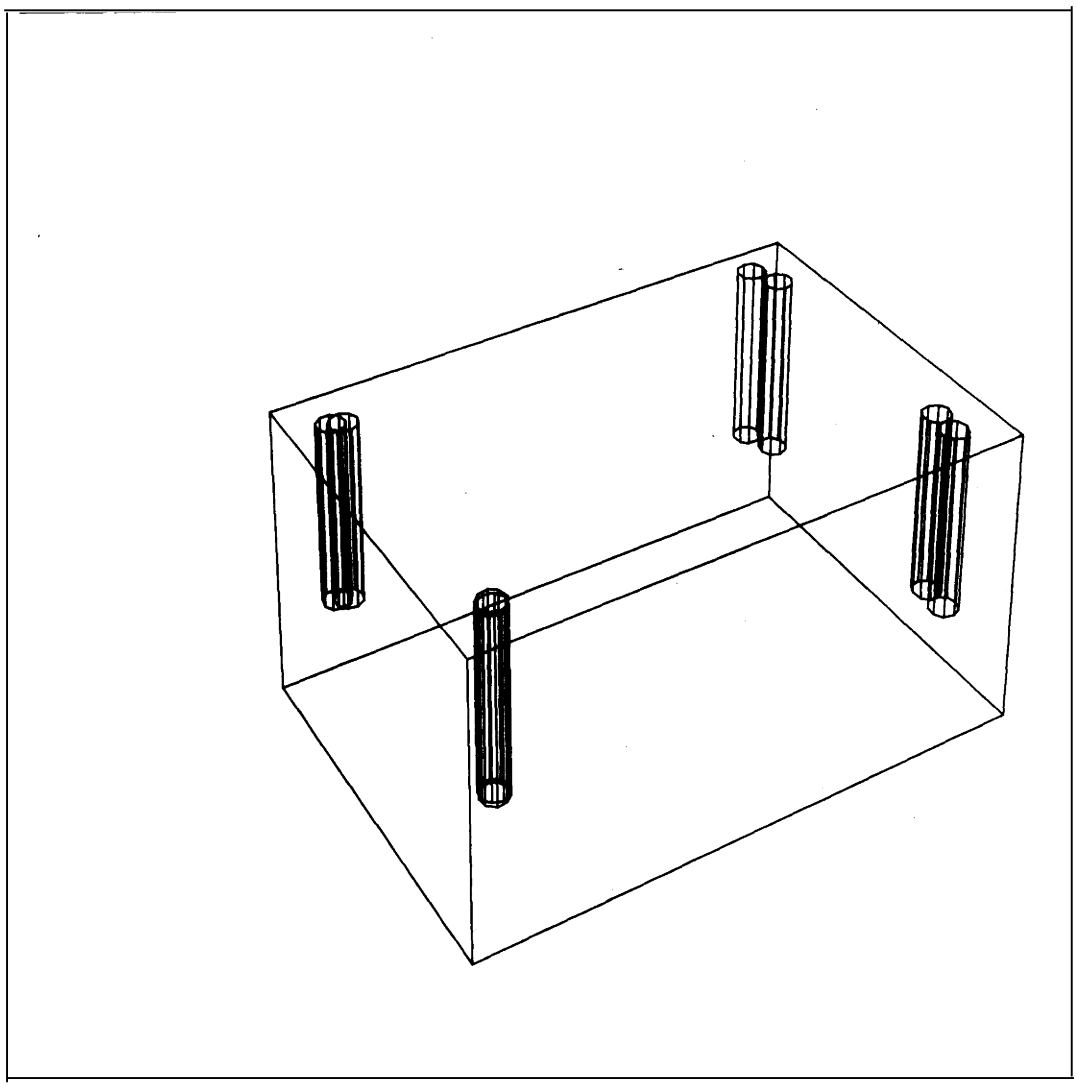


Figure 3: Double-Exposure Drawing of Rectangular Box

The PGMS routine ORIGIN returns the origin vector associated with the frame of the object pointed to by the first argument. Finally, the locations of each of the four holes in an ensemble of 500 boxes may be printed by attaching these semantics and executing the RECTBOX.

```
DECLARE (POS1(3),POS2(3),POS3(3),POS4(3)) FLOAT;
DO I=1 TO 500;
    CALL BEGIN(80000);
    CALL EXEC(HOLFIND);
    CALL SOLID(RECTBOX,12,8,4,          /*X,Y,Z*/
               2.5,0.3,1,               /*LENG,RAD,NSECT*/
               GAUSS(0.1/3),            /*XERR*/
               GAUSS(0.1/3),            /*YERR*/
               RAND(-2.5,2.5),          /*ANGERR*/
               GAUSS(0.05/3));          /*RADERR*/
    CALL END;
    PUT SKIP DATA (POS1,POS2,POS3,POS4);
    END;
```

Execution time is about 8 minutes on an IBM 370/168. A "double-exposure" drawing showing overlapping views of two boxes in the ensemble appears in Figure 3. This drawing was generated by attaching a standard graphics semantic routine and calling the RECTBOX procedure twice. The fact that graphics are produced so easily within PGMS is of considerable help in verifying that the simulation is working properly.

One aspect of this simulation which is perhaps unrealistic is that the fixture is perturbed for each box in the ensemble. In an actual manufacturing Operation, on the other hand, the fixture would be locked in place. The statistical distributions obtained in the actual manufacturing operation would therefore be narrower than those derived from this simulation.

What has been simulated here is an ensemble of boxes produced by *independent* setups as opposed to an ensemble produced by a *fixed* setup. In most cases of batch production, this simulation would be good enough for all practical purposes. One can imagine situations, however, in which the independent setup assumption is not appropriate. For instance, if pairs of consecutive boxes were to be attached to one another, the fact that both were produced on the same setup might be important. For this case, the code would have to be changed to simulate pairs of boxes instead of single boxes.

Actually, the box would probably be manufactured by trying a succession of setups until one was found which yielded satisfactory boxes, and this setup would then be retained for the remainder of the batch. Simulating the resulting ensemble is possible within PGMS, but it entails modeling the conditions used to determine whether or not the setup is satisfactory, Modeling conditional decisions is discussed briefly in the section of this paper dealing with extensions of the Monte Carlo method.

Box and Lid Assembly

This example is concerned with attaching a lid to the box of the previous example. The lid is 12 cm by 8 cm by 0.5 cm thick and is assumed to have been manufactured in the same manner as the box. At assembly time, a fixture is used which holds the lid rigidly in place on top of the box in such a way that the edges line up perfectly. The issue is whether or not the holes in the lid are aligned sufficiently well with those in the box to allow four screws to be inserted.

A procedure which represents both the box and its lid is shown below.

```
BOXNLID: ENTRY (X,Y,ZBOX,ZLID,LENG,RAD,NSECT,
                XERRB,YERRB,ANGERRB,RADERRB,
                XERRL,YERRL,ANGERRL,RADERRL);
CALL SOLID(RECTBOX,X,Y,ZBOX,LENG,RAD,NSECT,        /*BOX*/
                XERRB,YERRB,ANGERRBB,RADERRB);
CALL ZTRAN(ZBOX);
CALL SOLID(RECTBOX,X,Y,ZLID,ZLID,RAD,NSECT,        /*LID*/
                XERRL,YERRL,ANGERRL,RADERRL);
RETURN;
```

The next step is to code a semantic routine which computes the alignment errors for each of the four pairs of holes.

```
ALIGNER: PROCEDURE (NODE);
DECLARE (BOX,LID) POINTER;
DECLARE NHOLE BINARY FIXED;
DECLARE NODE ENTRY;
IF NODE-BOXNLID THEN DO;
     FOR NHOLE=1 TO 4 DO;
          CALL DEFINE (BOX,1,NHOLE+1,1);
          CA LL DEFINE (LID,2,NHOLE+1,1);
          ERROR(NHOLE)=DISTOZ(BOX,LID);
          END;
     END;
RETURN;
END ALIGNER;
```

The subaddresses in these DEFINE statements identify frames of corresponding CYLNDR holes in the box and lid. The function DISTOZ returns the distance from the Origin to Z-axis (OZ) of these two frames.

If it is assumed that the assembly process is unsuccessful whenever any of the four screw hole misalignments exceeds 2 mm, a simple procedure can be written to determine the number of successful assemblies in an ensemble of 500 boxes and lids.

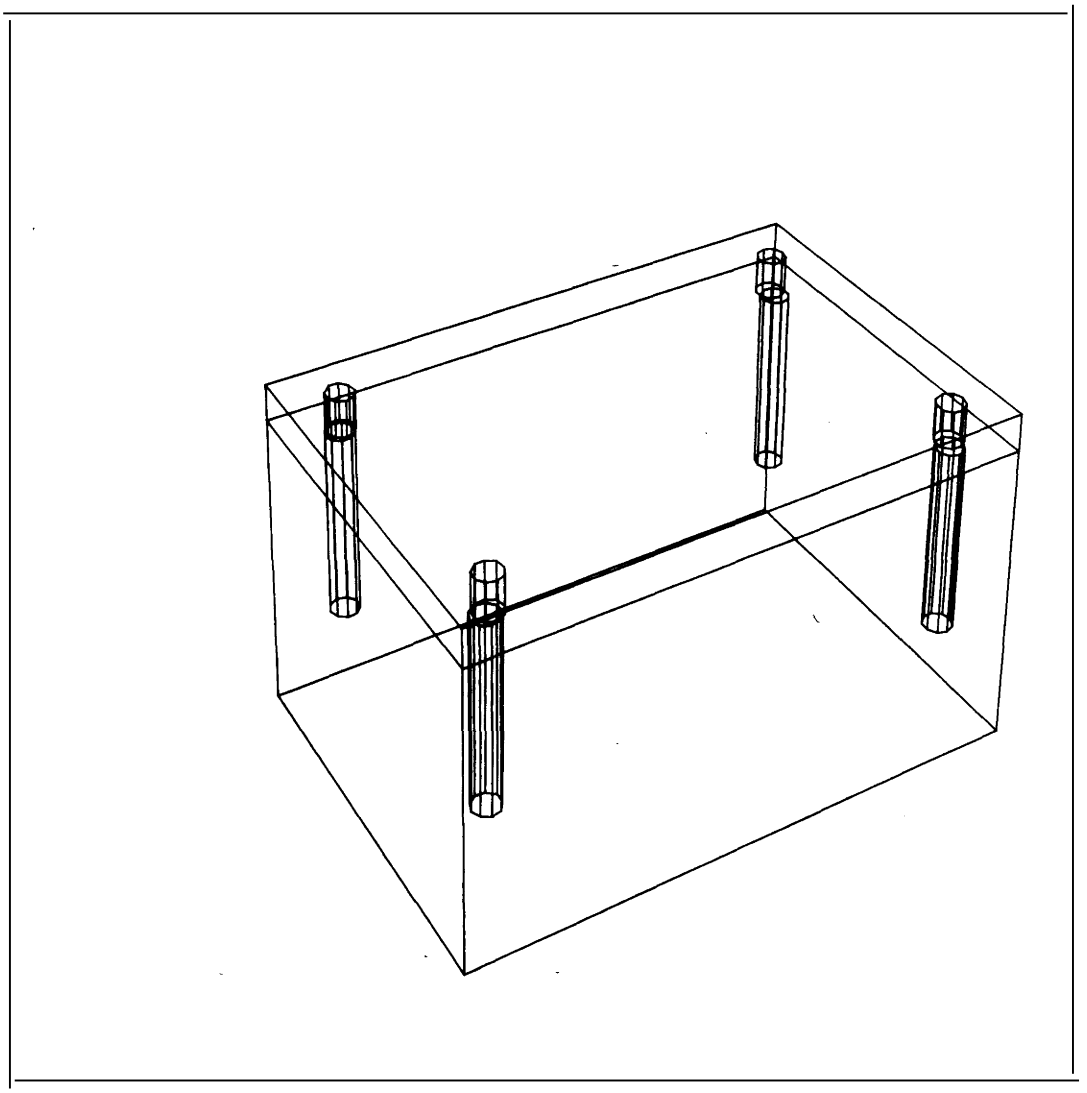


Figure 4: Drawing of Unsuccessful Box and Lid Assembly

```
DECLARE ERROR(4);
DO I-1 TO 500;
    NSUCCESS-0;
    CALL BEGIN( 120000);
    CALL EXEC(ALIGNER);
    CALL SOLID(BOXNLID,12,8,4,0.5,      /*X,Y,ZBOX,ZLID*/
               2.5,0.3,1,               /*LENG,RAD,NSECT*/
               GAUSS(0.1/3),            /*XERRB*/
               GAUSS(0.1/3),            /*YERRB*/
               RAND(-2.5,2.5),          /*ANGERRB*/
               GAUSS(0.05/3),           /*RADERRB*/
               GAUSS(0.1/3),            /*XERRL*/
               GAUSS(0.1/3),            /*YERRL*/
               RAND(-2.5,2.5),          /*ANGERRL*/
               GAUSS(0.05/3));          /*RADERRL*/
    CALL END;
    IF ERROR(1)<.2
        & ERROR(2)<.2
        & ERROR(3)<.2
        & ERROR(4)<.2
        THEN NSUCCESS=NSUCCESS+ 1;
    END;
PUT SKIP DATA (NSUCCESS);
```

When this program is executed, it determines that 27% of the assemblies would be successful. About 10 minutes of CPU time are required to obtain this result using an IBM 370/168. A drawing of one of the unsuccessful assemblies is shown in Figure 4.

Since in principle the lids are symmetric, it is also possible to generate an ensemble in which the lids have been randomly flipped upside down or rotated 180 degrees in the horizontal plane between the time of manufacture and the time of assembly. Such an ensemble simulates the common industrial practise of throwing freshly manufactured parts into a tote bin. The simulation then yields 19% successful assemblies. The reason why this percentage is much lower than the previous one is related- to the fact that the rotational error in the fixture was assumed to be about an axis which ran through a corner of the box rather than through its center.

Stanford Arm

The final example is taken from the field of computer controlled manipulators. Currently, two manipulator arms are being used at the Stanford University Artificial Intelligence Laboratory to study problems in industrial automation. Figure 5 shows a drawing of one of these arms holding a power screwdriver and a screw. Although the arm had been modeled much earlier by Baumgart,[16] this picture was obtained by using PGMS procedures instead.

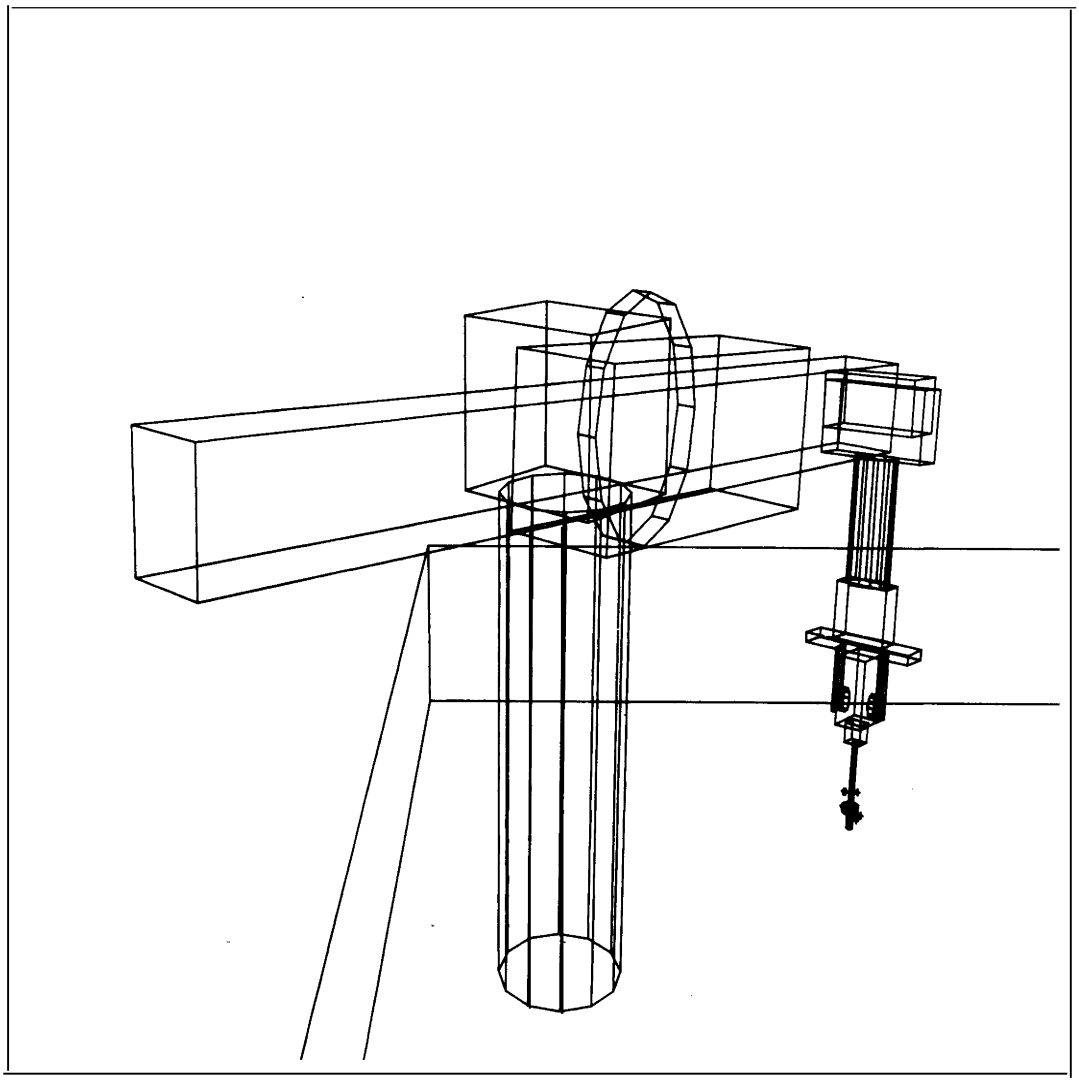


Figure 5: Drawing of Stanford Arm

In advanced manipulator applications, it is frequently necessary to perform *inspection* to measure the locations of objects or even simply to determine whether an object is present or missing. For instance, since a screw Can easily fall Off a screwdriver, it may be desirable to verify that the screw is actually still on the end of the screwdriver.

Both touch sensing and computer vision have been used in the past to perform this type of inspection.[20] Currently, Bolles is working on a more systematic approach to doing inspection by computer vision.[21] One of the main problems encountered in this endeavor relates to the fact that the location of the end of the screwdriver is not known precisely by the program, because of backlash and compliance in the manipulator. The vision program, therefore, can not simply look at the nominal location of the screw. Instead, it must search the image over a finite region *whose* extent depends on the tolerance errors of the manipulator joints.

The purpose of this example is to show that it is possible to do a Monte Carlo simulation of as complex an object as a manipulator, without having to write down the trigonometric formulas for the location of its gripper as a function Of all the joint angles. An ensemble of 10 Stanford arms may be modeled simply by coding

```
DO I=1 TO 10;
    CALL SOLID(SUARM,-41+RAND(-2,2), /*JOINT ANGLE 1*/
                    -92+RAND(-2,2),  /*JOINT ANGLE 2*/
                    15+RAND(-.2,.2), /*JOINT ANGLE 3*/
                    -90+RAND(-2,2),  /*JOINT ANGLE 4*/
                    90+RAND(-2,2),   /*JOINT ANGLE 5*/
                    0+RAND(-2,2),    /*JOINT ANGLE 6*/
                    1.5);            /*GRIPPER OPENING*/
    END;
```

It is only slightly more difficult in PCMS to model an ensemble of arms, each of which is holding a screwdriver with a screw. A semantic routine may then be supplied to draw the first object in this ensemble, and for all subsequent objects to draw a little asterisk at the location of the tip of the screw, as shown in Figure 5. Alternatively, semantics may be provided to compute the parameters of an error ellipse in the image plane, so that a vision program will know what region must be searched to verify the presence of the screw.

EXTENSIONS

In all four of the preceding examples, the simulation of tolerancing was used to derive *independent* distributions of resultant properties. It is also possible to derive *conditional* distributions of resultant properties. The need for considering conditional distributions arises primarily whenever there are steps in the manufacturing and assembly process which

involve conditional actions. Actually, such actions are quite common in discrete parts production, although they tend to be overlooked because these steps are usually implicitly assumed.

For instance, one expects an assembly worker 'to know without being told that

IF the lid doesn't fit
THEN throw it out and try another one
ELSE attach it

Alternatively, the worker might ignore any rquirement of interchangeability and save the nonfitting lid until a matching box was found. In either case, the statistical properties of the resulting assemblies would no longer be the same. This fact is true whether or not the conditional instructions are stated explicitly.

Not all conditional actions have the simple form IF . . . THEN . . . ELSE. For example, the assembly process might involve sliding the lid until it is aligned with the box. This step would move each lid by a different amount, depending on the initial misalignment of that particular lid and box.

In a PGMS tolerancing simulation, the addition of steps which simulate conditional actions is a straightforward process, provided that these actions can be stated in the form of procedures which involve spatial transformations no worse than rotations and translations by well defined amounts. For an IF . . . THEN . . . ELSE action, one simply adds the appropriate IF ... THEN . . . ELSE clause to the program. A problem arises, however, that there ate conditional actions which can not be easily expressed in the form of well defined procedures.

A common and insidious example of such actions relates to the way parts are often chamfered to make the assembly process easier. As the assembly is performed, the chamfers force parts into slightly different positions and alter their subsequent statistical properties. The effect of a chamfer in locating a single pin can be expressed fairly easily in the form of a 'procedure, but for more than one pin the effect of chamfering becomes very difficult to state explicitly.

The effect of chamfers is a specific case of a general process which may be called fitting or *accommodation*. Case studies performed at the Charles Stark Draper Laboratory indicate that in typical industrial assemblies, roughly 15% of the steps involve accommodation.[22] Although this process is industrially important, it is very difficult to simulate except in the simplest situations. For instance, it is well known that the way to attach a lid to a box is to put all four screws in loosely and then tighten them, rather than tightening each one immediately. Unfortunately, even in this case it is not known how to express the exact process of accommodation in the form of a well defined procedure.

However, it is possible to *approximate* many accommodation processes. For example, in the

box assembly one can say that the first screw to be loosely inserted produces a translation of the lid such that its hole aligns with the corresponding box hole. The second screw produces a rotationof the lid which makes the vector from the first to the second lid hole align with the corresponding box vector, followed by a translation of the lid along this vector to make the two alignment errors equal and opposite. The third screw only produces a translation orthogonal to the previous vector, and the fourth screw has no effect. Clearly, this procedure is only an approximation to what really happens, but the chances are that it is a good enough approximation for most practical purposes. An alternative approximation would be to say that each successive screw produces a transformation of the lid to a new position such that the sum of the squares of of the alignment errors is minimized. In either of these cases, one can easily add to PGMS procedures which simulate the approximate accommodation process.

Another extension of Monte Carlo tokrancing would be to simulate the process of making measurements with imperfect measuring tools. For example, suppose a computer vision system is used to locate the position of a hole in a part so that a manipulator can insert a screw. This measurement is limited by the camera resolution, Which may be on the order of one picture element in the scanning array. The measurement is also limited by pan and tilt errors in aiming the camera. Projecting the camera errors from the image plane back to the actual hole in three-dimensional space will generally give an elongated region within which the location of the hole can not be resolved. If several features of a part are located in this manner, the position and orientation of the part itself may be derived. All of these steps can be simulated within PGMS.

It is also possible to simulate part imperfections of a much grosser nature than those normally considered in tolerancing. For instance, Agin has written a computer vision program 'which inspects lamp bases for displaced or *missing* grommets.[23], In 'order to simulate an ensemble of lamp bases with an appropriate range of defects, one could represent the generic lamp base by a routine with parameters specifying whether or not the grommets are present.

```
LAMPBAS: ENTRY (GROM 1,X 1,Y 1,GROM2,X2,Y2);
CALL XYZTRAN(X 1,Y 1,0);
IF GROM1=1
     THEN CALL SOLID(GROMMET);
CALL XYZTRAN(X2-X 1,Y2-Y 1,0);
IF   GROM2=1  ,
     THEN CALL SOLID(GROMMET);
RETURN;
```

Gross defects of this type are quite common in industry. The most familiar example is that roughly 2% of all machine screws are ordinarily defective. Some have no heads, while others have no slots or no threads. The defective fraction may be reduced by preinspection, but for most applications the additional cost can not be justified. It is therefore worth emphasizing the fact that errors of these types can also be simulated within a Monte Carlo parts

tolerancing system.

CONCLUSION

This paper has described a Monte Carlo approach to the simulation of tolerancing and other forms of imprecision in discrete parts manufacturing and assembly. An implementation of the method, based on the Procedural Geometric Modeling System developed earlier by this author, is illustrated by four specific examples, one Of which was chosen from the field of assembly by computer controlled manipulators.

There appears to be a pressing need for simulation techniques relating to discrete parts manufacturing and assembly. The assembly process is strongly affected by imprecise components, imperfect fixtures and tools, and inexact measurements. It is often necessary to design higher precision into the manufacturing and assembly process than is functionally needed in the final product. Production costs are highly dependent on specified tolerances and the resultant product yields.

The technique described in this paper can provide production engineers with a systematic way of analyzing the stochastic implications of tolerancing and other forms of imprecision.

ACKNOWLEDGEMENT

This paper was partially motivated by Russell Taylor's work on high level languages for computer controlled manipulators. One of his programs determines allowed loci of workpieces by resolving symbolic geometric constraints. The paper was also motivated by the computer vision research of Bob Bolles. One of his programs calculates the region of an image to be searched for a desired feature of a workpiece that has been displaced slightly from its nominal position. Discussions with Taylor and Bolles in the early stages of this work have proved very valuable. Their results, incidentally, will be published soon as part of their doctoral dissertations.

This work was performed at the Stanford AI Lab, as part of the Computer Integrated Assembly Systems project headed by Tom Binford. I want to thank Peter Will, manager of the Automation Research project at the IBM T. J. Watson Research Center, from which I was on sabbatical leave, for making my year at SAIL possible.

Finally, I want to acknowledge the logistical assistance of Mike Blasgen and Larry Lieberman in this work.

REFERENCES

[1] W. V. Tipping, An *Introduction to Mechanical Assembly*, Business Books, London, England, 1969.

[2] *An Introduction* to *PADL*, Production Automation Project Technical Memorandum TM-22, University of Rochester, December 1974.

[3] A. A. G. Requicha, N.M. Samuel, and H. B. Voelcker, *Part and Assembly Description Languages - II*, Production Automation Technical Memorandum TM-20a, University of Rochester, revised November 1974.

[4] *Discrete Part Manufacturing: Theory and Practice*, Production Automation Project Technical Report TR-l-1, University of Rochester, 1974.

[5] *Dimensioning and Tolerancing*, American National Standards Institute Report ANSI Y 14.5- 1973, published by IEEE, New York, 1973.

[6] Lowell W. Foster, *Geometric Dimensioning and Tolerancing: A Working Guide*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1970.

[7] Earlwood T. Fortini, *Dimensioning For Interchangeable Manufacture*, Industrial Press Inc., New York, 1967.

[8] Harold W. Gugel, *Monte Carlo Simulation With Interactive Graphics*, GM Research Publication CMR-1531, General Motors Corporation Research Laboratories, Warren, Michigan, October 1974.

[9] John M. Hammersley and David C. Handscomb, *Monte Carlo Methods*, Wiley, New York, 1964.

[10] Gerald J. Agin, *Representation and Description of Curved Objects*, Stanford Artificial Intelligence Laboratory Memo AIM-173 and Stanford University Computer Science Report STA N-CS-305, October 1972.

[1 1] Gerald J. Agin and Thomas 0. Binford, *Computer Description of Curved Objects*, Third International Joint Conference on Artificial Intelligence, Stanford, August 1973.

[12] Ramakant Nevatia, *Structured Descriptions of Complex Curved Objects for Recognition and Visual* Memory, Stanford Artificial Intelligence Laboratory Memo *AIM.250* and Stanford University Computer Science Report STAN-CS-464, October 1974.

[13] I. C. Braid, *Designing With Volumes*, Cantab Press, Cambridge, England, 1974.

[14] I. C. Braid, *The Synthesis of Solids Bounded by Many Faces,* Communications of the ACM, Volume 18, Number 4, p. 209, April 1975.

[15] Bruce G. Baumgart, *Winged Edge Polyhedron Representation,* Stanford Artificial Intelligence Laboratory Memo AIM- 179 and Stanford University Computer Science Report STAN-CS-320, October 1972.

[16] Bruce C. Baumgart, *GEOMED,* Stanford Artificial Intelligence Laboratory Memo AIM-232 and Stanford University Computer Science Report STAN-CS-4 14, May 1974.

[17] David D. Grossman, *Procedural Representation of Three-Dimensional Objects,* IBM Research Report RC-5314, T. J. Watson Research Center, Yorktown Heights, N. Y., March 14, 1975; to be published in IBM Journal of Research and Development.

[18] Mark A. Lavin and Laurence I. Lieberman, *A System for Modeling Three-Dimensional Objects,* IBM Research Report RC-5765, T. J. Watson Research Center, Yorktown Heights, N. Y., December 17, 1975.

[19] Mark A. Lavin, *MODFEAT: A Sysfcm for Naming Polyhedral Features of Three-Dimensional Objects,* IBM Research Report RC-5764, T. J. Watson Research Center, Yorktown Heights, N. Y., December 17, 1975.

[20] Robert Bolles and Richard Paul, *The Use of Sensory Feedback in a Programmable Assembly System,* Stanford Artificial Intelligence Laboratory Memo AIM-220 and Stanford University Computer Science Report STAN-CS-396, October 1973.

[21] Robert C. Bolles, *Verification Vision Within a Programmable Assembly System: An Introductory Discussion,* Stanford Artificial Intelligence Laboratory Memo AIM-275 and Stanford University Computer Science Report STAN-CS-75537, December 1975.

[22] J. Nevins, D. Whitney, S. Drake, D. Killoran, M. Lynch, D. Seltzer, S. Simunovic, R. M. Spencer, P. Watson, and A. Woodin, *Exploratory Research in Industrial Modular Assembly,* Charles Stark Draper Laboratory Report R-921, Cambridge, Massachusetts, December 1, 1974 to August 31, 1975.

[23] C. Rosen, D. Nitzan, G. Agin, G. Andeen, J. Berger, J. Eckerle, G. Gleason, J. Hill, J, Kremers, B. Meyer, W. Park, and *A. Sword, Exploratory Research in Advanced Automation,* Stanford Research Institute Project 2591 Report 2, Menlo Park, California, August 1974.