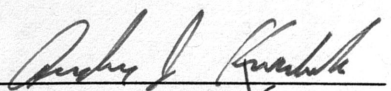
Video Processing Approach to Control of
Large Space Structures

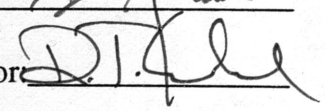
Laurrie Ann Wittig
University Undergraduate Fellow, 1990-91

Dr. A.J. Kurdila
Assistant Professor
Department of Aerospace Engineering

Texas A&M University

APPROVED:

Fellows Advisor 

Honors Program Director 

CONTENTS

I. INTRODUCTION

The Department of Aerospace Engineering has been successful in the past few years in establishing an analytical and experimental capability in the area of space structure dynamics and control. Still, based upon recent topics targeted for investigation at the EVTEK (Technology for Space Station Evolution, March 1990) workshop by NASA, it is desirable that the Spacecraft Dynamics and Control Laboratory at Texas A&M University further broaden the spectrum of its dynamics and control experiments. Specifically the following key research areas have been identified as enabling technologies for the attitude and structural control of the forthcoming space station

1. Control sensitivity due to inertia variance of the space station during construction. Because the space station will require control strategies appropriate for differing configurations as it undergoes construction, fixed control strategies that are either robust with respect to changes in the system inertia, or adaptive to changes in system inertia developed.
2. Control strategies for Nonlinear Dynamics. During operation of the onboard manipulator system on the space station, the governing equation of dynamics are not only inertia variant, but nonlinear due to large displacements and highly flexible nature of the system. Control strategies that account for these nonlinear dynamics must be developed and experimentally verified before construction of the space station can be confidently undertaken.
3. Evolutionary character of the space station. In addition to construction, the evolution of the space station into a transportation node for future space flights will drastically alter its configuration and system inertial properties as shuttles dock, and additional modules are added.

For these reasons, a nonlinear dynamics and control experiment is sought to be developed for the Spacecraft Dynamics and Control Laboratory. This experiment should be sufficiently general in nature and to be applicable to a variety of specific experiments intended to study implementation issues associated with robust control theories/methodologies for systems that are inertia variant in time, are highly flexible, can change configuration, and have nonlinear governing dynamics.

Because it is intended that the hardware be designed to be general and suitable for a wide collection of individual experiments, the goals of this overall research are far-reaching and broad in scope. Practically speaking it is anticipated that the author will not accomplish all the goals listed below. Still, the project is designed to be "open-ended".

1. The foremost goal of the research is to design a prototype multiflexible-body experiment for use with a wide range of control strategies.
2. After completion of the final design of the nonlinear multibody dynamics experiment, the hardware will be implemented in the Spacecraft Dynamics and Control Laboratory.
3. Concurrently with phase (2) above, feasibility studies are conducted employing optical sensing and approximation theory to learn the inverse kinematics of the experiment.

(II) HARDWARE DESCRIPTION

Figure 1 depicts the overall features of the experiment. The base truss structure consists of a combination of rigid aluminum tubes (lengths 8 inches and 12 inches) connected by aluminum nodes. The tubes and nodes are manufactured by Meroform. In the current configuration, the base truss has a height of 53 inches, width of 42 inches, and an approximate weight of 45 pounds. The structure is also counterbalanced to reduce undesired moments about the x-axis. This specific configuration is based on ASTREX (Advanced Space Structures Experiment) at Edward's Air Force Base illustrated in Figure 2.

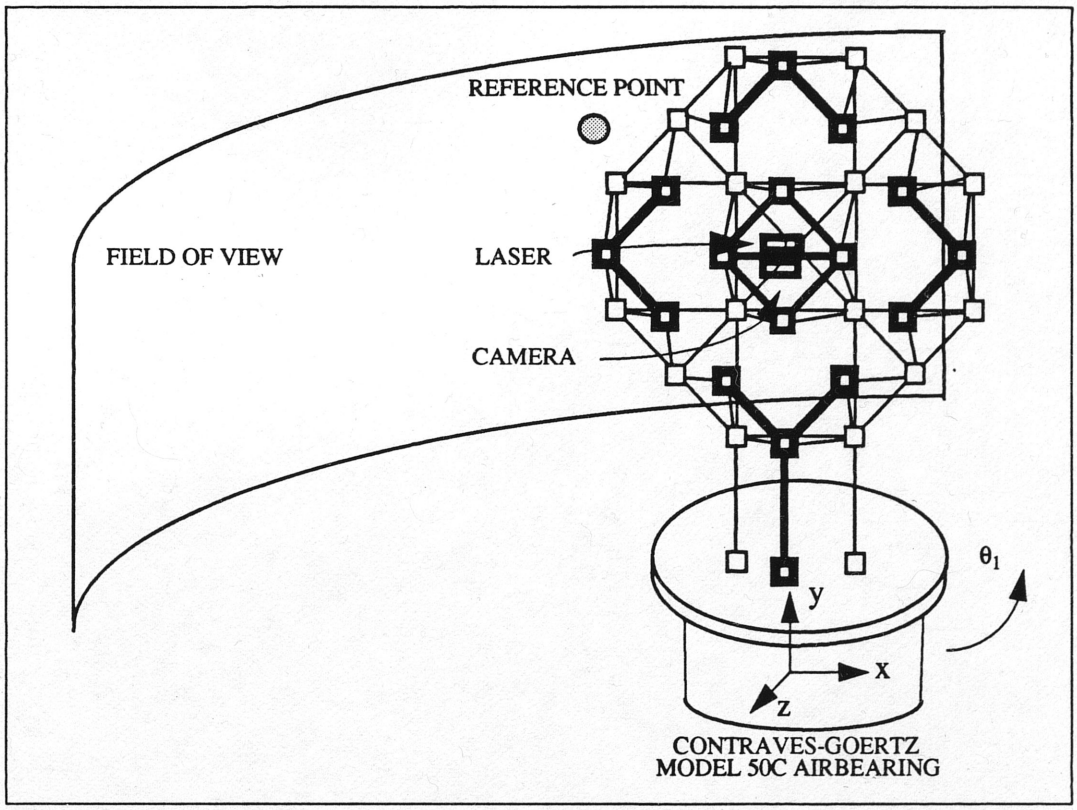


Figure 1. Experimental Configuration.

The truss structure is mounted to the table top surface of the Contraves-Goertz Model 50C Air Bearing incorporating one degree of freedom, θ_1, into the experiment. The air bearing offers rotation with extremely low friction and high angular precision (<1/3 arc second accuracy). Control is attained via Model 30H Modular Precision Angular Control System (MPACS) either locally by an on board processor or through a high CPU interface by directly programing the control and status registers. The latter was used for data collection.

Mounted to the center aluminum tube on the base structure is a NEC TI23A 512 x 512 CCD camera below a Uniphase Model 1500 Series Helium-Neon Laser. The camera is operable at 30 Hz/frame (60 Hz/field) with standard RS-170 output data format. Located at the tip of the camera lens is a Motion Analysis Mac Light LED (Light Emitting Diode) ring. The laser was not used in the present work, however, it will be implemented in future experiments.

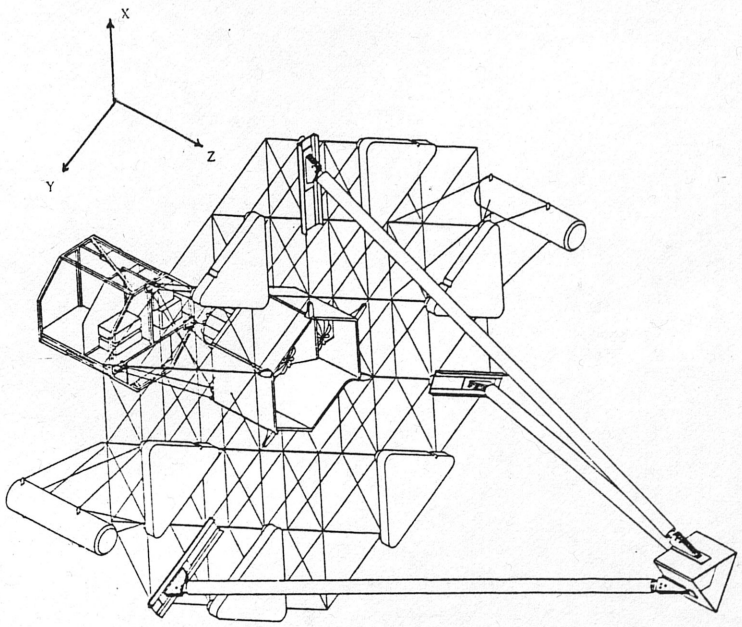


Figure 2. ASTREX (Advanced Space Structures Experiment).

The image acquisition hardware consists of the Data Translation DT2861 Arithmetic Frame Grabber and the DT7020 32-bit Floating Point Array Processor. Communication between the units is accomplished by two 8-bit "highways" (DT-connect) which supports high speed (≅ 1MHz) synchronous image transfers between the boards. The DT2861 consists of an 8-bit flash A/D converter which digitizes and stores a 512 x 512 (512 lines or rows by 512 pixel or columns per line) frame in real time (30 Hz). The image is stored as pixels in a one dimensional array as illustrated in Figure 3. Each pixel is assigned a grey scale value between 0 and 255 depending on its intensity. Once the image has been acquired, windowing or region manipulations may be implemented to examine a specific area of interest. Other on board supporting functions include arithmetic, logical, and statistical operations along with frame averaging. After transferring the data to the DT7020 array processor via DT-connect, the image may be processed independent of either the host PC or the DT2861 Frame Grabber. The array processor then passes the results to the host processor for interpretations. The DT2861 contains 4 Mbytes of on board, dual-ported memory organized as sixteen 512x512x8-bit frame storage buffers, while the DT7020 contains 3 Mbytes RAM for application and 1Mbyte for system memory.

Both the DT2861 and DT7020 are controlled from a host IBM PC/AT 286 five star computer by C-callable routines. The Frame Grabber may be programmed directly by addressing the control and status registers, however, the present work utilized the DTIRIS subroutine library for acquisition and frame manipulations. On board processing of the DT7020 was accomplished by calling a "handle" address corresponding to a macro sequence loaded to the array processor prior to program execution. Memory allocation and initialization of the image acquisition hardware occurred before program execution, allowing for real time image acquisition and processing.

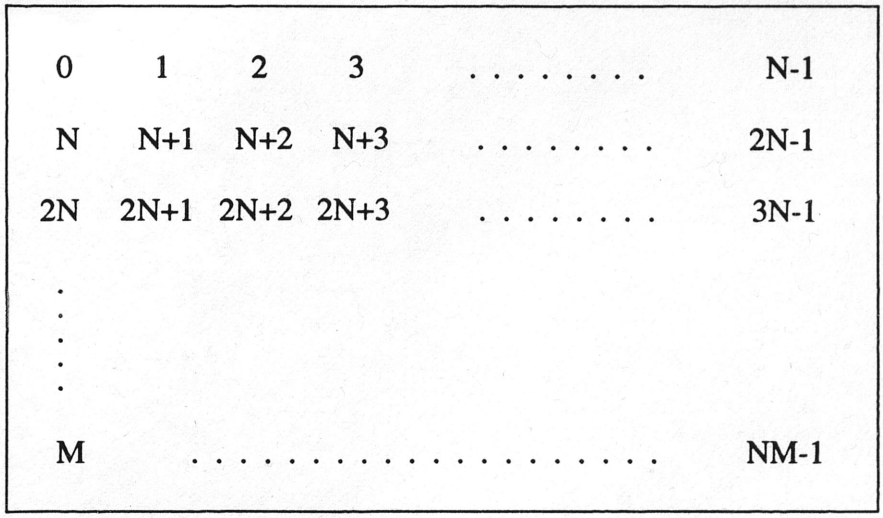


Figure 3. Frame Grabber Pixel Storage Illustration as a One Dimensional Array for Frame of size NxM.

The field of view consists of 1/8 inch pexi-glass tensioned to a semicircle at a distance eight feet from the air bearing. The field of view is four feet in height and elevated two feet above the ground. Mounted 6 3/4 inches from the top of the pexi-glass is a one inch diameter "dot" of tape centered horizontally which serves as the reference point. The tape is manufactured by 3M (7610 High Gain Sheeting) and exhibits a luminesce factor of 900x when the LED ring light shines perpendicular to the point and 700x with the LED ring light is skewed by 45 degrees. However, even with the decrease in the luminesce factor at a skewed angles, the pixel response was not significantly effected.

A summary of the system components along with arrows illustrating data paths are illustrated in Figure 4. Note that the host PC serves as the primary controller for the equipment and may operate independent of the DT2861 and DT7020.

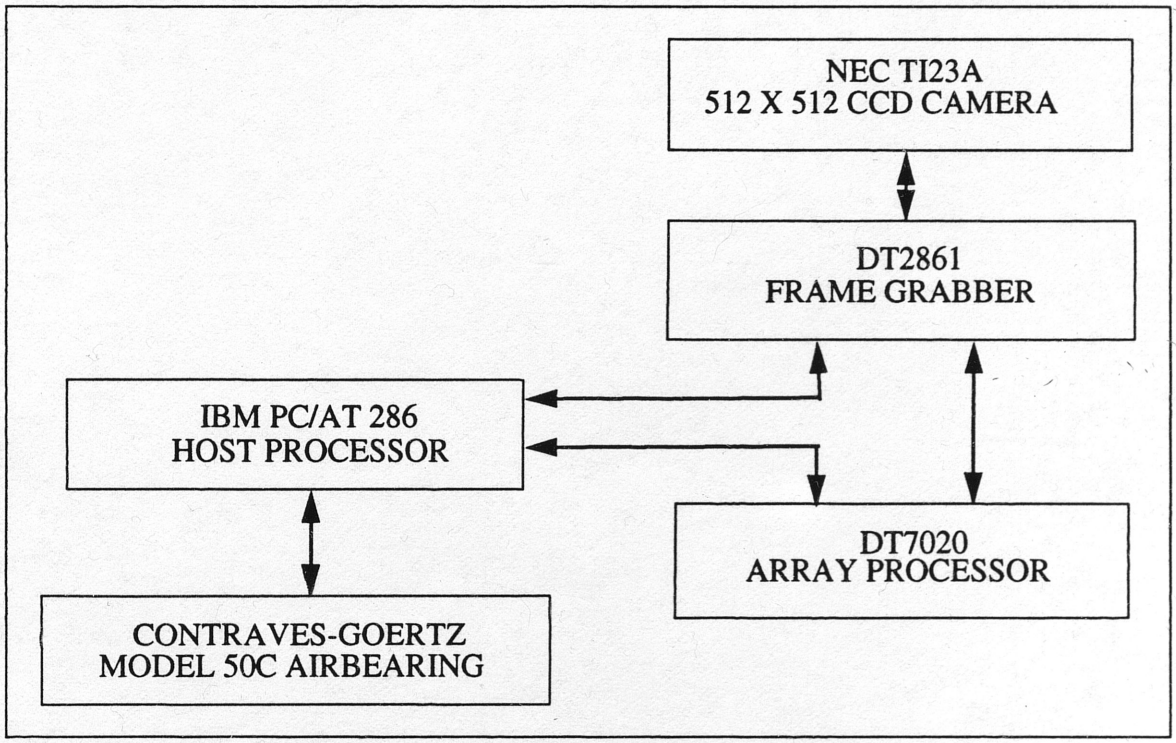


Figure 4. Control/Image Acquisition Flowchart.

(III) HARDWARE CONTROL

The experimental process employed to collect data included positioning the camera to a specific angular location, acquiring a digitized frame, determining the center pixel location of the reference point, then printing the pixel location and corresponding angles to a data file. This process was repeated for one hundred different angular positions at a constant angular increment. As illustrated in Figure 4, the equipment used for this procedure includes the Contraves-Goertz Air Bearing, DT2861 Frame Grabber, and DT7020 Array Processor with the host PC serving as the central controller.

Contraves-Goertz Air Bearing

Only one degree of freedom exists in the current configuration, the azimuth - θ_1, incorporated by the air bearing. Three methods exist to control the air bearing: active address, matrix control method, and line ID control method. The attached air bearing programs, ABPOS.C and ABRATE.C, utilize the line ID method. With this, several functions may be controlled by a single data word by writing the data word to a specific address. Two MetraByte cards (24 bit high output current parallel digital interface, Model PIO-24) were used for this interfacing. Their addresses and control word configurations, defined at the beginning of each program, are illustrated in Figure 5 by hexadecimal integers. Note the output and input ports are relative to the computer. After the cards are configured, the air bearing is ready to accept commands.

Signal requirements for the air bearing are defined as "0" for true and "1" for false; opposite from normal convention. Therefore, all commands sent to the air bearing use the bitwise not operator (~).

The line ID control method incorporates a two word transfer. First the mode command is established by addressing octal 0101 to Port C, Card 1. Next, command the mode read by addressing octal 005 to Port C, Card 1. In each case (mode command and mode read), ports A and B of card 2 are assigned to the desired mode, i.e., position mode, precision rate mode, or off mode.

Now that the mode has been chosen, the air bearing is ready to accept or give specific positions or rotation values. Again this is accomplished via two word transfer--coarse resolution and fine resolution data words addressed successively to port C of Card 1. (All octal addresses used in attached programs are defined by MPACS, therefore the bitwise not operator is omitted.)

Beginning with the coarse resolution command, two 8-bit hexadecimal integers corresponding to the coarse resolution of the angular input are addressed to ports A and B of Card 2. This is repeated for the fine resolution; two 8-bit hexadecimal integers corresponding to the fine resolution of the angular input are addressed to ports A and B of Card 2. The subroutines poscom() in attached program ABPOS.C and prerate() in attached program ABRATE.C illustrate this portion of the control sequence. To incorporate reading the position or precision rate rotation, ports A and B of card 1 (input ports) are read after the coarse and fine resolution reading modes are addressed to port C of card 1.

The basic procedure to follow in programing the Contraves-Goertz Air Bearing is illustrated by a flowchart in Figure 6.

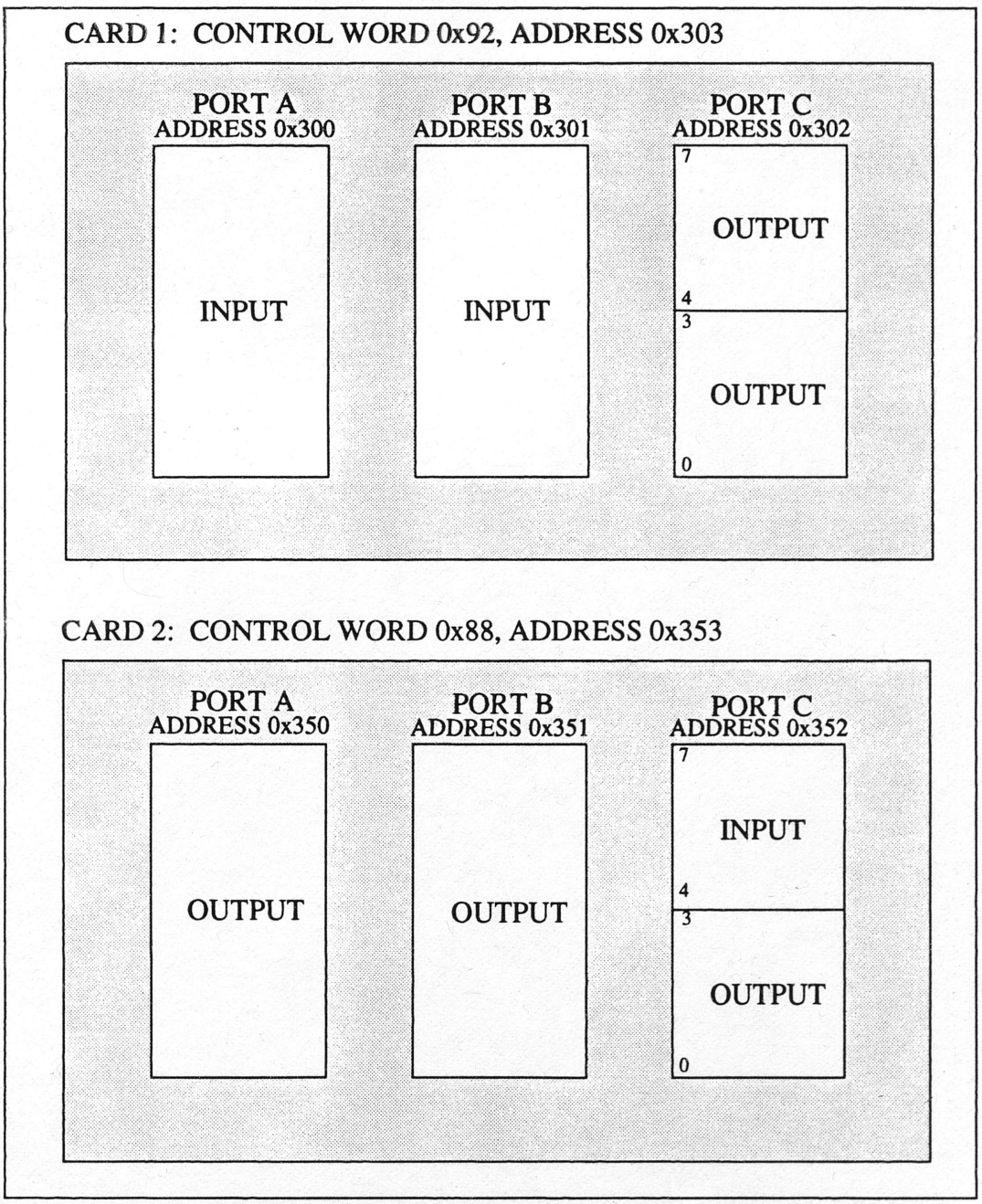


Figure 5. Configuration of MetraByte Cards used for Interfacing with Contraves-Goertz Air Bearing.

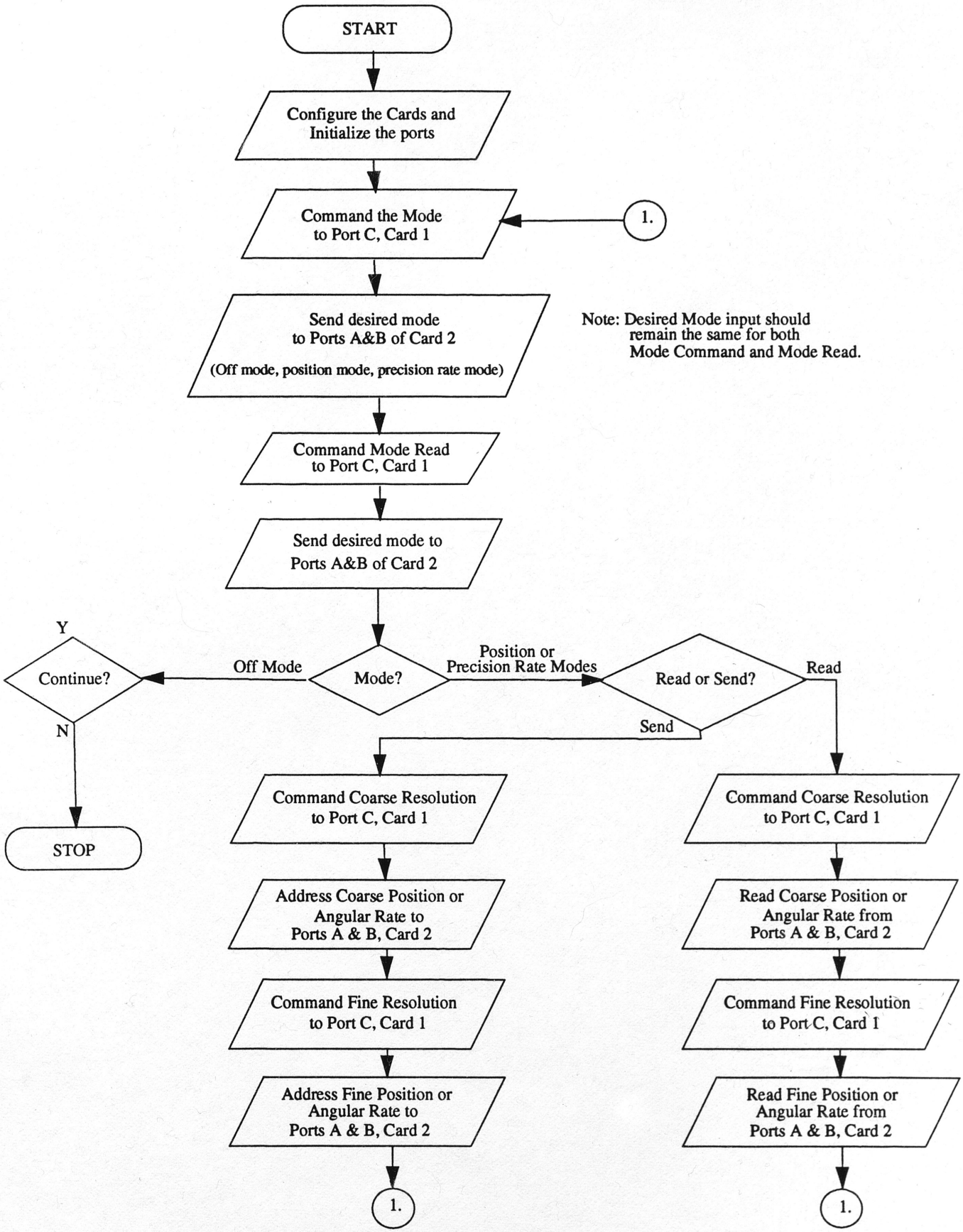


Figure 6. Flowchart demonstrating control of the Contraves-Goertz Air Bearing for position, precision rate, and off modes.

DT2861 Frame Grabber

As previously mentioned, the frame grabber may be controlled by directly programming the control and status registers, however, the attached programs utilize a C-callable subroutine library -- DTIRIS.

The basic programming sequence illustrated by FRAME.C for acquiring data is listed below. Note all subroutines defined by DTIRIS begin with "IS_".

1. Initialization. This must be the first step for Frame Grabber usage. This command opens a channel to DTIRIS, allocates memory, and resets all registers to the default state.
2. Input Look Up Tables. The input look up table is an array of size 256, numbered 0 to 255, corresponding to the possible pixel grey scale values. The pixel value serves as an index into the table, and the value at that location is stored in the frame store memory as the pixel value. DTIRIS contains seven preprogrammed input look up tables that may be called throughout the program, or, an input look up table may be programmed and loaded to the frame grabber for use.
3. Resultant Look Up Tables. Although not used in FRAME.C, the resultant look up table is an array of size 512, numbered 0 to 511. After a frame has passed through the input look up table, it may be passed to the frame grabber arithmetic logic unit (ALU) for computation. The size of this array corresponds to the possible values output from the ALU. Again, the new pixel value serves as an index into the table and the value at that location is stored in memory. DTIRIS contains three preprogrammed resultant look up tables that may be called through out the program, or, a resultant look up table may be programmed and loaded to the frame grabber memory for use.
4. Output Look Up Tables.Similiar to the input look up table, the output look up table is an array of size 256 corresponding to the possible pixel grey scale values. Here, the data to be displayed is transformed through the output look up table, exhibiting the intensity corresponding to the stored value in the table. As with the input look up tables, DTIRIS contains seven preprogrammed output look up tables, or an output look up table may be programmed and loaded to frame grabber memory, as with the other look up tables.
5. Display Programming. In this section, options are selected which are used for acquiring a frame. For example, selecting the frame buffer for data storage, turning display on, setting the sync state to external (which must be used for acquiring images), and clearing frame memory. Also, an active region may be set at this time or after the data acquisition.
6. Video Input. The program FRAME.C utilized the DTIRIS command "IS_ACQUIRE(-frame number, frame count)" for this purpose. This subroutine basically acquires the number of frames specified by frame count, averages the frames, then stores the data in the buffer assigned by frame number.
7. External Port Output. This utilizes the DT-connect to transfer data from the camera to the array processor. FRAME.C does not incorporate this command; however, it is used in UNIFORM.C, the program that incorporates the use of the air bearing, frame grabber, and array processor.
8. Cease Frame Grabber Operations. This is accomplished by calling the DTIRIS subroutine "IS_END()".

FRAME.C was used in determining proper windowing regions and pixel resolutions. A 48 x 60 pixel frame was chosen centering the reference point both vertically and horizontally. This windowing technique would allow for a more rapid determination of the center pixel location and reduce required memory storage aboard the array processor. Also, the location would be more accurate due to the reduction of noise by omitting part of the frame.

After determining a window region, the pixel resolution was determined. The current configuration is such that the center pixel will notice a one pixel change for an angular increment of 0.03 degrees.

DT7020 Array Processor

The array processor was incorporated into the experiment for the purposes of high speed vector calculations, specifically, to process a centroiding technique used to calculate the center pixel location of the frame.

In the present case, the center pixel location was determined by an intensity weighted centroiding scheme using the expressions:

$$x = \frac{\sum_{i,j} x_{ij} I_{ij}}{\sum_{i,j} I_{ij}} \qquad y = \frac{\sum_{i,j} y_{ij} I_{ij}}{\sum_{i,j} I_{ij}}$$

where

(x,y) = centroid location image of array

I_{ij} = grey scale intensity at $(i,j)^{th}$ pixel

(x_{ij},y_{ij}) = position of the $(i,j)^{th}$ pixel

The array processor calculates the first moments of the array about a specific pixel. The summed values are then passed to host memory for final determination of the center pixel location.

The MACH DSP Subroutine Library provides the high-level access to the DT7020. Mach DSP is a software package consisting of C-callable routines. These subroutines allow performance of high speed data acquisition and processing. One limitation is the 3 Mbytes available data storage capacity. These basic steps should be followed when programming the array processor:

1. Initialize the array processor.
2. Allocate memory space aboard array processor.
3. Learn routine sequences. Define a code sequence (Macro), addressed by a "handle"
4. Run routine sequences. Each sequence is activated as the defined "handle" is called.
5. Collect and output results.
6. Deallocate Memory.
7. May repeat to sequence beginning at 2.
8. Stop the DT7020.

The attached program APPIXEL.C illustrates usage of the array processor. It is used to determine the centroid location of a dummy array formatted similar to input from the frame grabber.

UNIFORM.C provides a combination of the previous programs using the air bearing, frame grabber, and array processor to perform the desired data acquisition. It is designed to perform the desired data acquisition for an inputted number of data points, with an angular increment of 0.8°. This increment was chosen to give reasonable reference point location distinction. During the data acquisition, the air bearing was set to perform the position slew at a rate of $\cong 21°$/sec.

The image acquisition phase -- acquiring a frame and determining its center pixel location -- required only 110 microseconds to perform. However, another real-time consideration is the 30 Hz/frame procurement.

(IV) HARDWARE STATUS

The current structural configuration will serve as the base framework for future experiments. Photographs of the current configuration and laboratory setup are shown in Figures 8 - 10.

The next step includes attaching four flexible aluminum appendages, as shown in Figure 11, then extending the camera and laser to the tip. This will be used for the purpose of investigating non-linear pointing properties. These aluminum appendages and attaching brackets are built and the will be added to the base structure shortly. Also envisioned is to incorporate a second axis-- the elevation, θ_2. This second axis would be mounted on the base structure to allow for camera positioning in both the azimuth and elevation angles. With this configuration, areas to be studied include structural deformations plus tracking and vibration suppression with a slew/point manuever.

Although the exact usage of the Uniphase Laser has not been determined, many possibilities exist. One in particular includes using the laser as a movable target to be tracked by the camera on the field of view.

These changes would only require moderate programming alterations and additions into the existing control codes. While the additions to the structure become more complex, the applicability to actual systems increases.

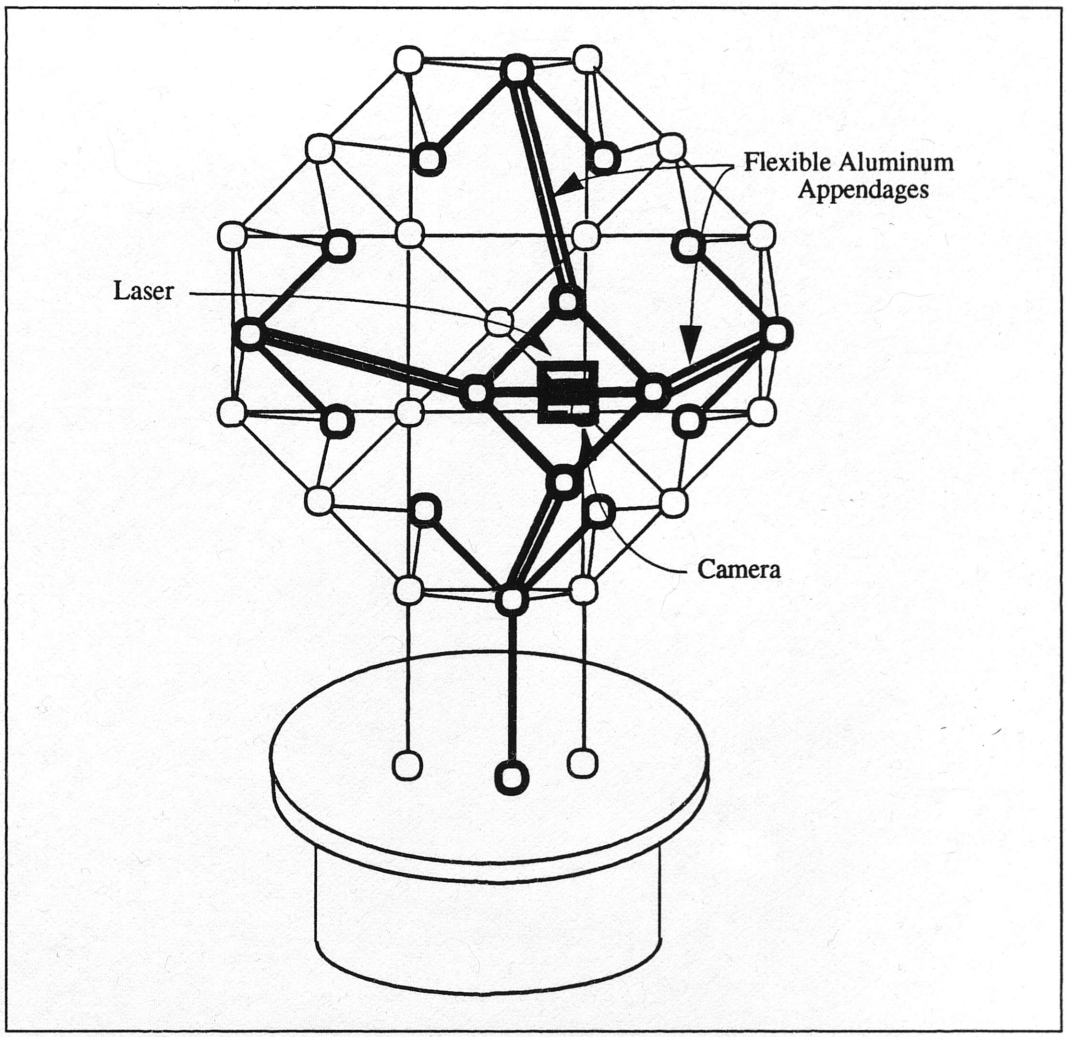


Figure 11. Future Configuration of Base Truss Structure.

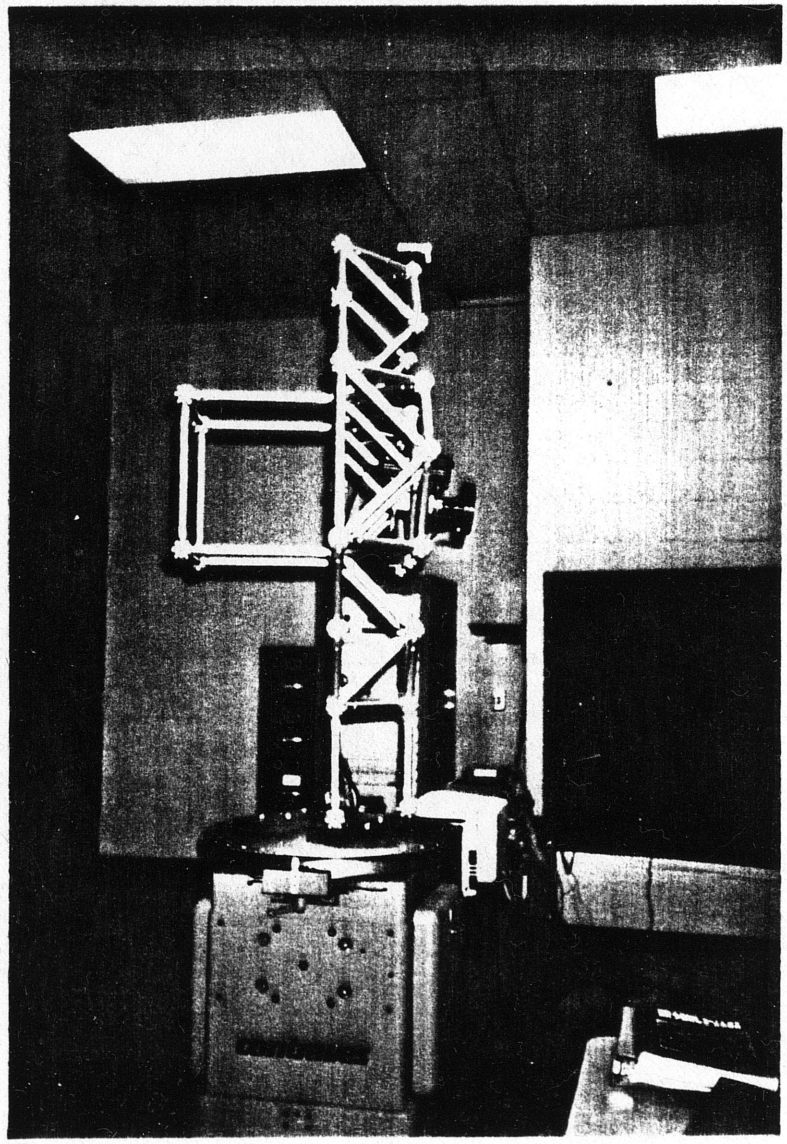


Figure 8. Side View of Base Truss Structure.

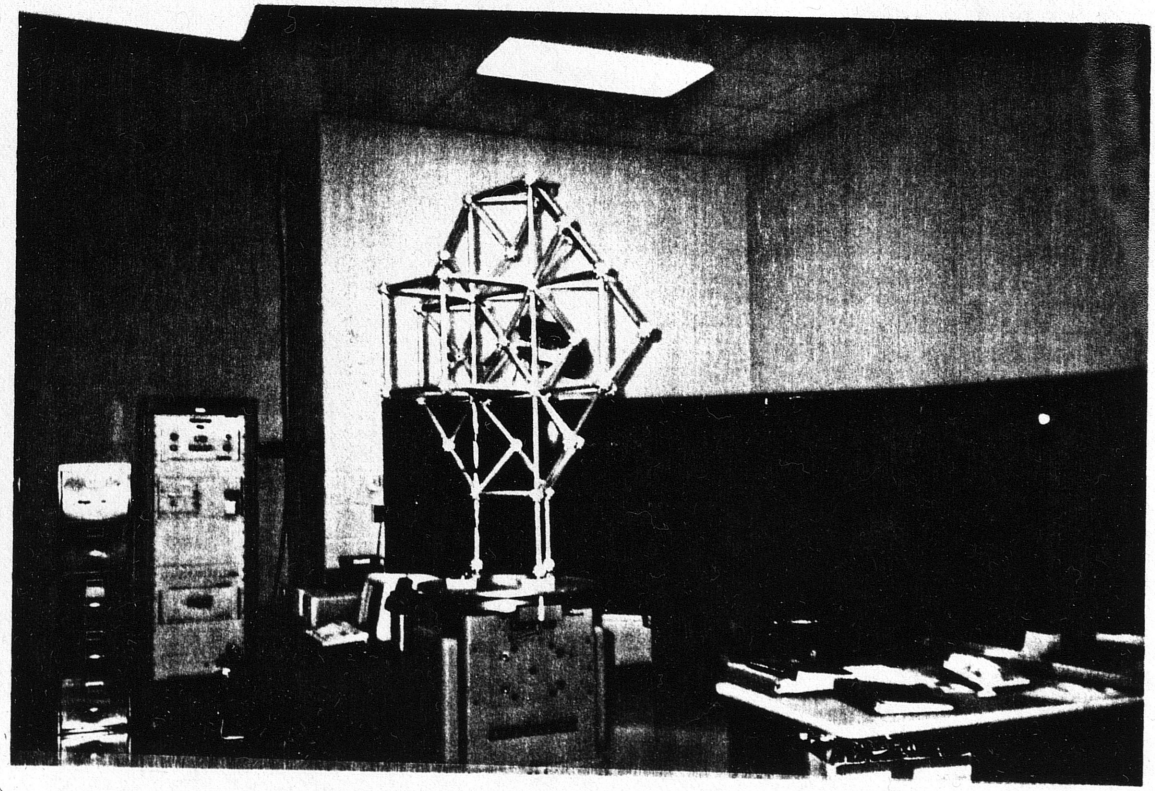


Figure 9. Current Laboratory Setup.

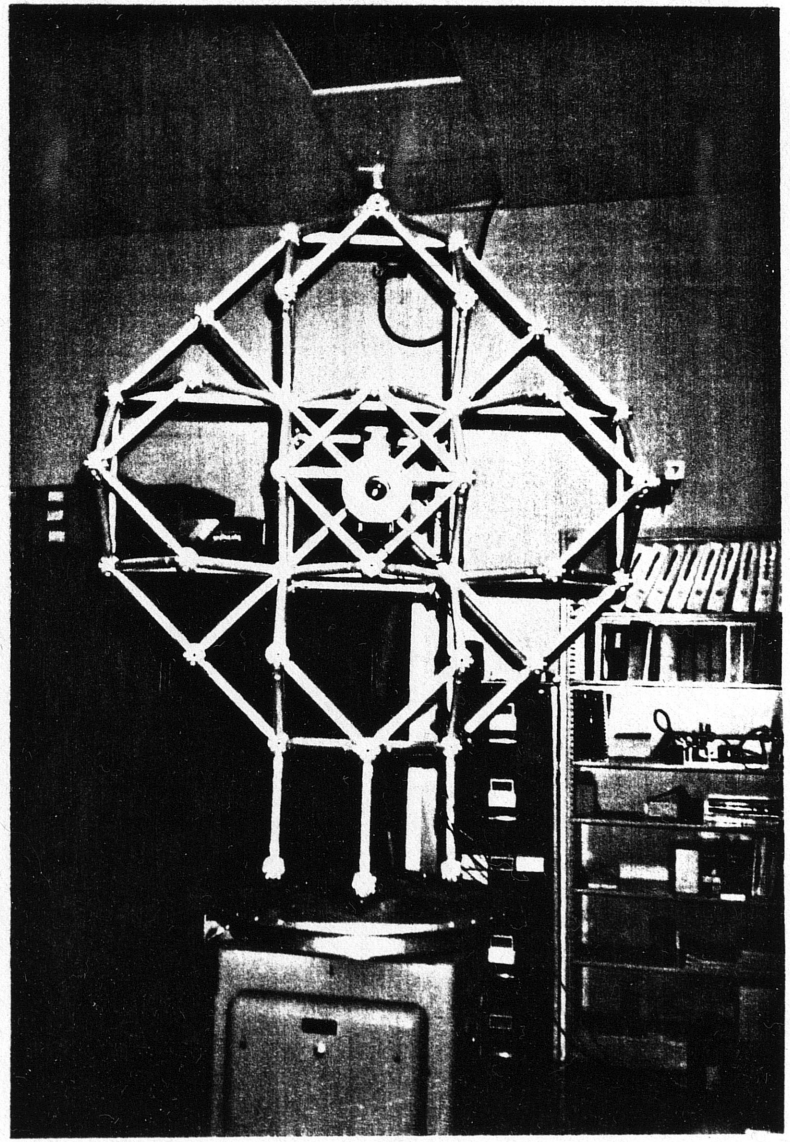


Figure 10. Forward View of Base Truss Structure.

(V) LEARNING ORIENTATION VIA VISION PROCESSING

While the design of the slew-and-point experiment has been carried out so as to be amenable to a number of various experiments dealing with

(i) line of sight pointing ,
(ii) trajectory tracking,
(iii) and vibration suppression,

the focus of this report is the investigation of image processing for the purpose of determining orientation. In a general sense, the work can be viewed as a study into the feasibility of using (near) real time vision processing to guide "proximity operations" involving the navigation of an autonomous craft in the vicinity of the space station.

This report specifically studies the feasibility of utilizing recent advances in the field neural networks to "learn" the input-output relationship between the image of an object retreived by the camera and the inertial coordinates of a specific object. As shall be discussed subsequently, the problem of "learning" the input-output relationship can be viewed int the context of approximation theory [Poggio,Girosi]. The literature on neural networks that has accumulated over the past few years is far to vast to consider in detail in this paper. Because of collateral interest by researchers at the Center for Approximation at Texas A&M [Ward, Narcowich,Pilant] in the field of radial basis function approximation, the approach taken in this paper follows their work, as well as that of [Poggio,Girosi].

The problem considered in this section involves the *supervised* learning of the input/output map that relates the camera geometry depicted in figure (5.1). The view depends upon the

(i) the camera image plane coordinates (x,y),
(ii) the object space coordinates (X,Y,Z),
(iii) the camera's orientation angles (Φ,Θ,Ψ),
(iv) the camera's principal point (X_c,Y_c,Z_c),
(v) the focal length f,
(vi) and the principal point offset (x_0,y_0).

Neural networks have been employed in technically similar problems of inverse kinematics [GF,-SLP] .The advantage in the present application is that complicated offline calibration methods such as resection [Anderson] could potentially be avoided using the method. The primary steps in the method are

(I) Data collection in real-time of (subsets of) data ($x,y,X,Y,Z,\phi,\theta,\psi$) while f, x_0, y_0, X_c Y_c, and Z_c are fixed.

(II) Neural learning to locate the centers (control points) for radial basis approximation.

(III) Solution of the radial basis approximation problem to identify the desired input/output relationship.

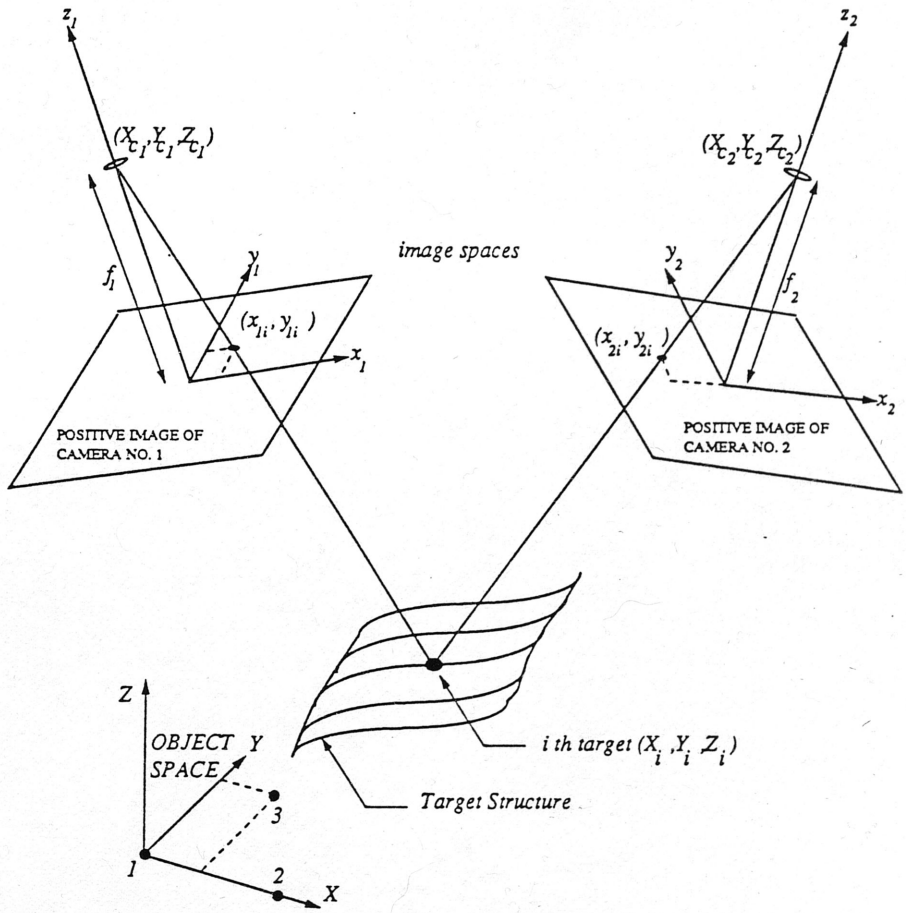


Figure 5.1 Stereo Triangulation Geometry

(VI) KOHONEN'S TOPOLOGY PRESERVING MAPPINGS

For the experiment at hand, it is possible, and quite likely that, huge amounts of data representing valid combinations of $(x,y,X,Y,Z,\phi,\theta,\psi)$ will be collected. Because of the variety of circumstances under which the pointing maneuver may be executed, this data may

(i) consist of literally hundreds or thousands of measurements,
(ii) be associated with disjoint areas in the camera's view,
(iii) contain regions of higher density where greater accuracy in the input/output model is sought.

If enough memory, and cpu time, is available, it is always possible to use one of the numerous methods available to interpolate all of the data, say using B-splines. However, one character of "learning" is that essential features of the the data at hand are selected to represent the group of data as a whole. In this section, one such method, Kohonen's Topology Preserving Mapping, is used to locate representative "centers" that reduce the amount of data considered in the final approximation.

A detailed presentation of Kohonen's Topology Preserving mapping and its ability to "self-organize" would require a great deal of rather technical discussion. Perhaps the simplest representation of Kohonen's network is given in pp 130-132 of [Kohonen]. In this presentation, one can visualize the neural network as receiving a set of scalar inputs

$$\xi = \{\xi_1 \ldots\ldots \xi_N\}^T \in \Re^N$$

and distributing these signals to an array of processing units

$$[\mu_{ij}] \in \Re^N$$

The output of the processing units are simply

$$\eta_i = \sum_j \mu_{ij} \xi_j$$

in this crude model as depicted in figure (6.1). With this notation, the goal of Kohonen's method can be stated rather succinctly:

"... to devise adaptive processes in which the parameters of all the (processing) units con-

verge to such values that every unit becomes specifically matched . . . to a particular domain of input signals. . ."

This statement can be clarified by the observatioin that if one denotes the rows

$$[\mu_{ij}] = \begin{bmatrix} m_1^T \\ \\ m_N^T \end{bmatrix} \qquad m_i^T = [\mu_{i1} \cdots \mu_{iN}]$$

then

$$\eta = m_i^T \xi$$

is a measure of the similarity between signal and row m_i^T -- it is just the inner product of vectors. Of course, this analogy can be extended only so far. The processor $\mu_{i,j}$ "adapt" as they are presented different input signals, and therefore can be viewed as functions of (descrete) time. In addition, the evolution, or adaptation, laws for learning are complicated, very among authors, and it is difficult to prove their convergence.

Fortunately, the algorithm describing the adaptation laws are quite simple. An example from [Kohonen] can be summarized as

(o) Choose a size of a "neighborhood of processors" N_c.

(i) Find the best similarity match for the input signal

$$\| \xi(t_k) - m_c(t_k) \| = \min_{i<N} \| \xi(t_k) - m_i(t_k) \|$$

(ii) Update all processor units in the neighborgood of the best match

$$m_i(t_{k+1}) = m_i(t_k) + \alpha(t_k) \{ \xi(t_k) - m_i(t_k) \}$$

(iii) Repeat for all input signals.

While the discussions thus far have been quite general there is a probabalistic interpretation of the weight vector s m_i that should be noted. If one defines the average quantization error

$$E = \int_{Vx} \| x - m_{c(x)} \|^r p(x)\, dV_x$$

where

$x \in \Re^N$	input vector
$dV_x \equiv$	volume differential
$c(x) \equiv$	index of closest weight vector to x
$p(x) \equiv$	point density of weight vectors

Then it can be shown that the algorithm above that " . . . the asymptotic values of the m_i . . . mini-

mize E" when $N_c = \{c\}$. In other words,

"the m_i cluster so as to minimize the probability of quantization error."

Figures (6.2) through (6.8) depict how Kohonen's method can be utilized to find "centers of approximation" for large collections of image data. In figures (6.2) through (6.5), the focal plane of a single point (1,1,1) in object space is shown as the camera is oriented at angles Φ and Θ that vary between 0 and .1 radian. In these cases, a 30 by 30 grid of image points have been simulated using the program COGEN.C located in the appendices. Instead of trying to utilized all 900 points of data, Kohonen's algorithm has been used to adaptively learn where the data is concentrated, and represents the data using a 10 by 10 grid. Starting from random locations, it is clear as the number of learning iterations increase from 100 in figure (6.2) to 25000 in figure (6.5), the approximation centers conform and distribute themselves over the data area. These results give a good visual interpretation of the "convergence in distribution" property of Kohonen's map discussed earlier. Figures (6.6) through (6.8) illustrate similar results, but as the angles Φ and Θ vary over 2 radians. The corresponding collection of image points is quite irregular in this case. However, the method produces reasonable results in as little as 12000 learning steps for this case. Similar results can be shown for *irregular disconnected regions*, which are discussed in subsequent sections.

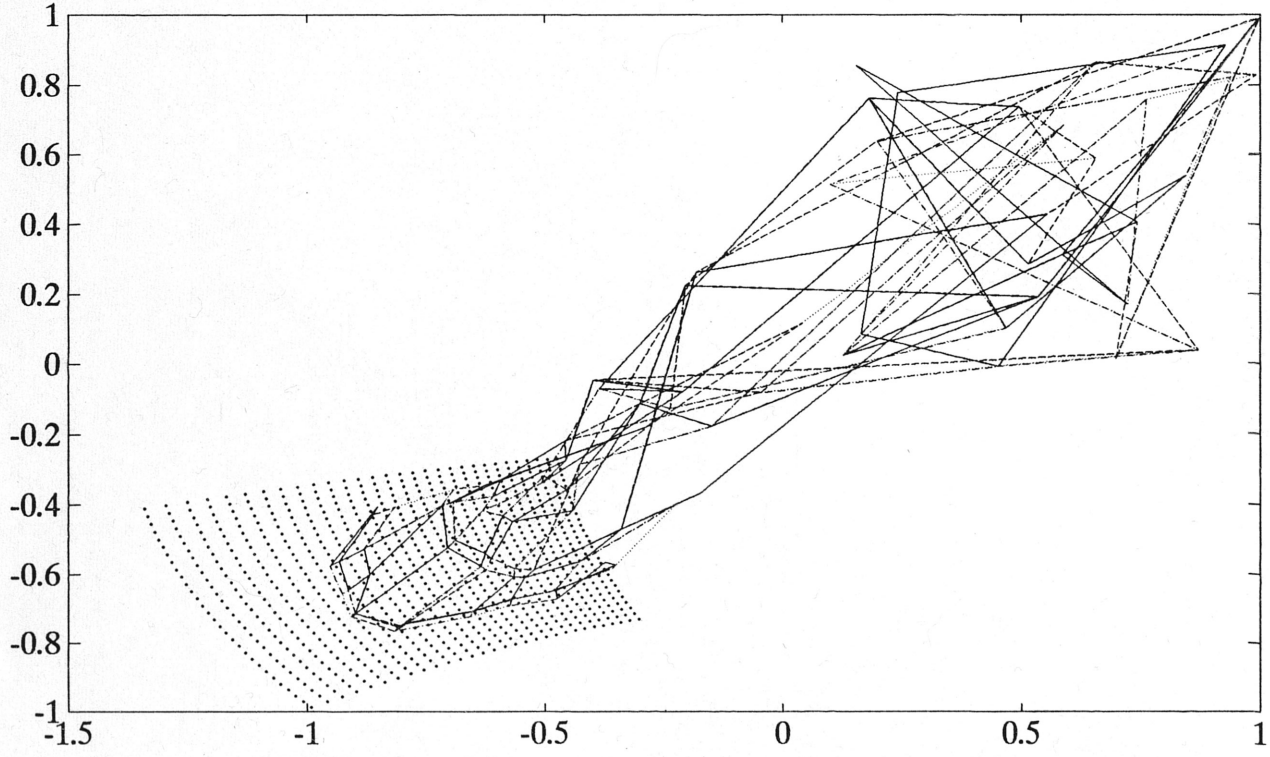


Figure (6.2) Kohonen's Algorithm : 100 iterations (<0.1 rad)

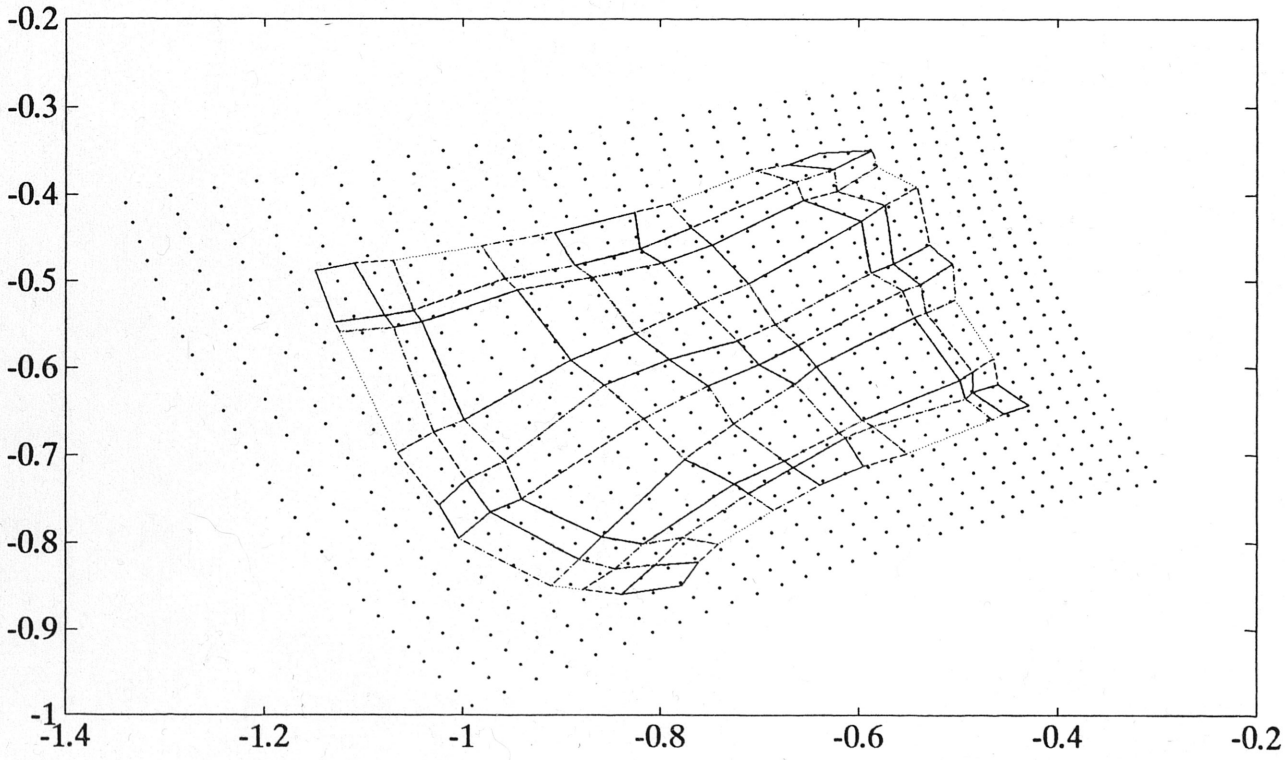


Figure (6.3) Kohonen's Algorithm : 1000 iterations (<0.1 rad)

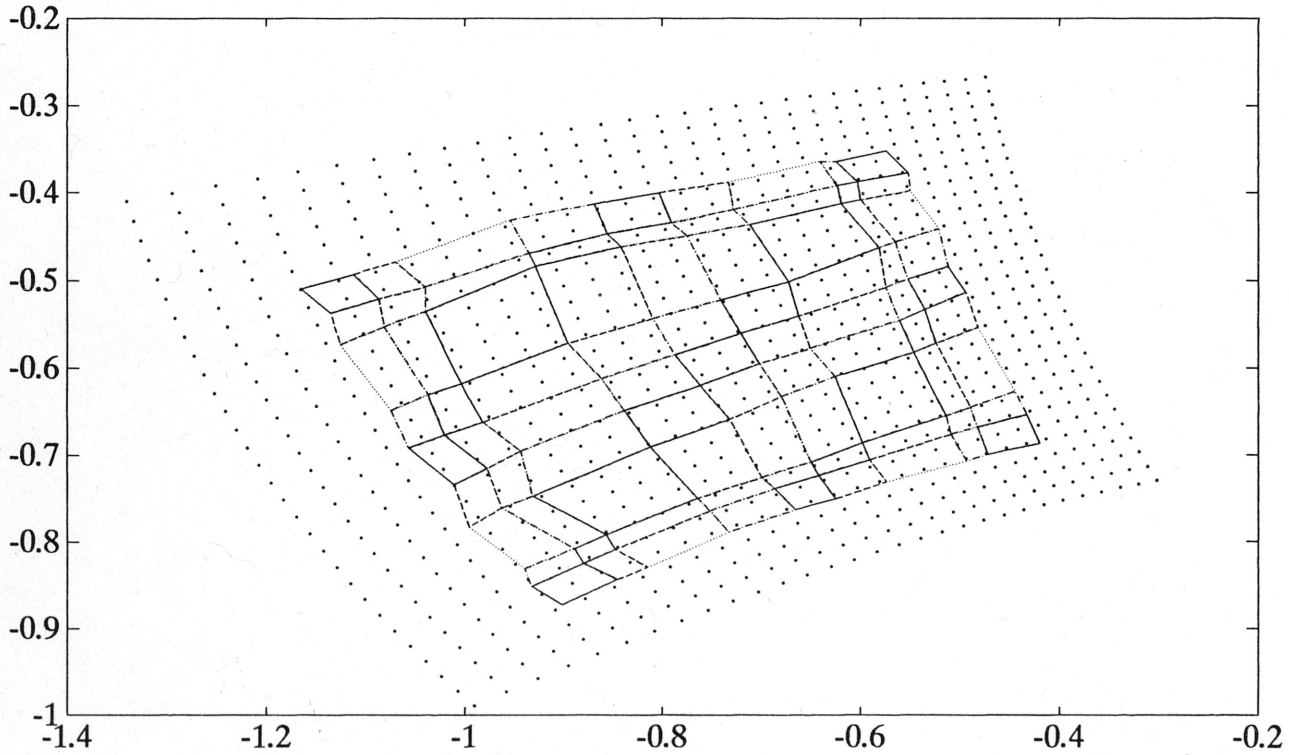


Figure (6.4) Kohonen's Algorithm : 5000 iterations (<0.1 rad)

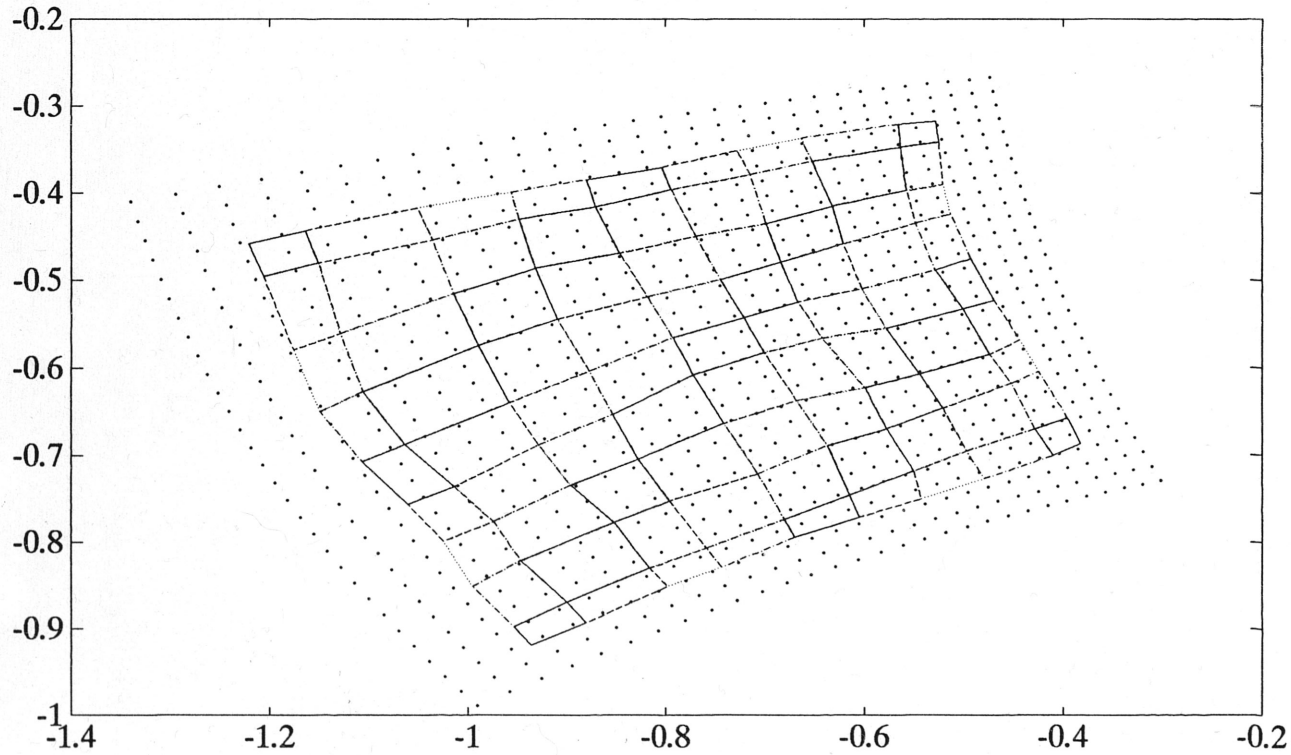


Figure (6.5) Kohonen's Algorithm : 25000 iterations (<0.1 rad)

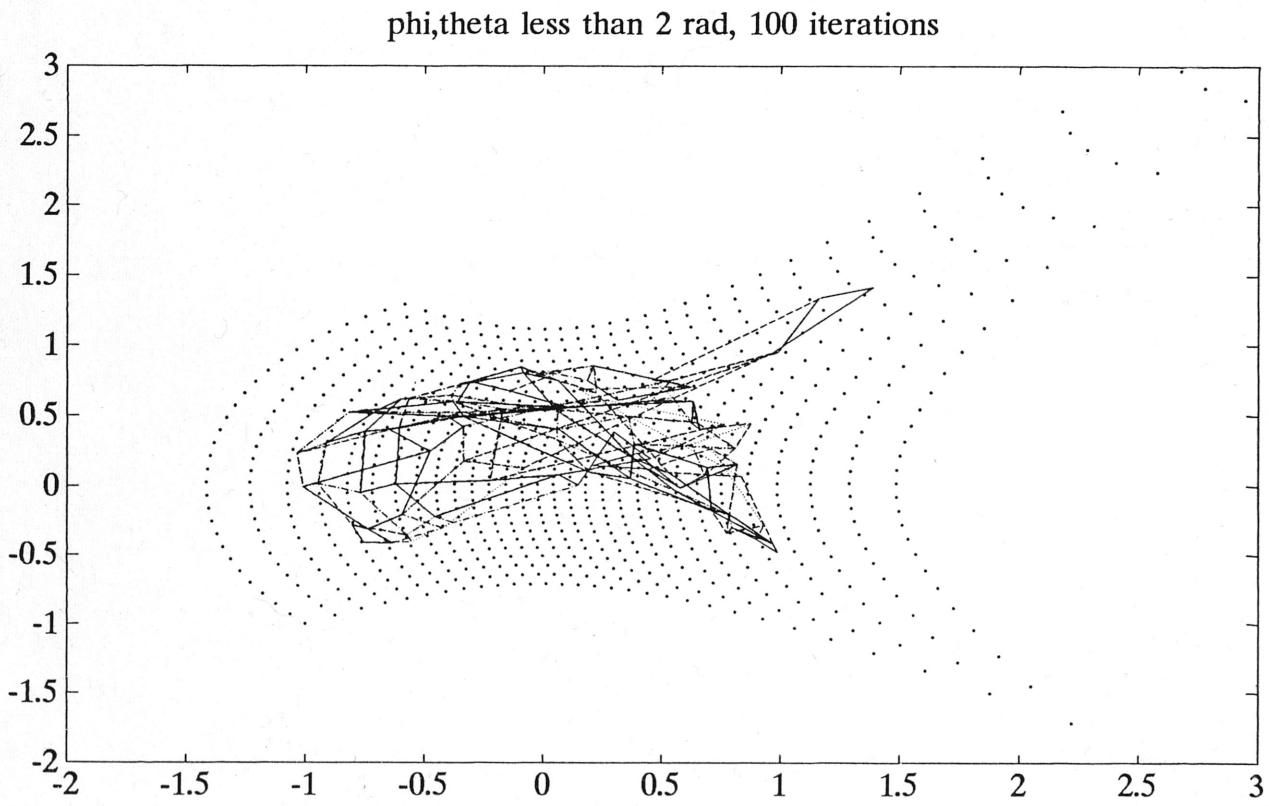


Figure (6.6) Kohonen's Algorithm : 100 iterations (<2 rad)

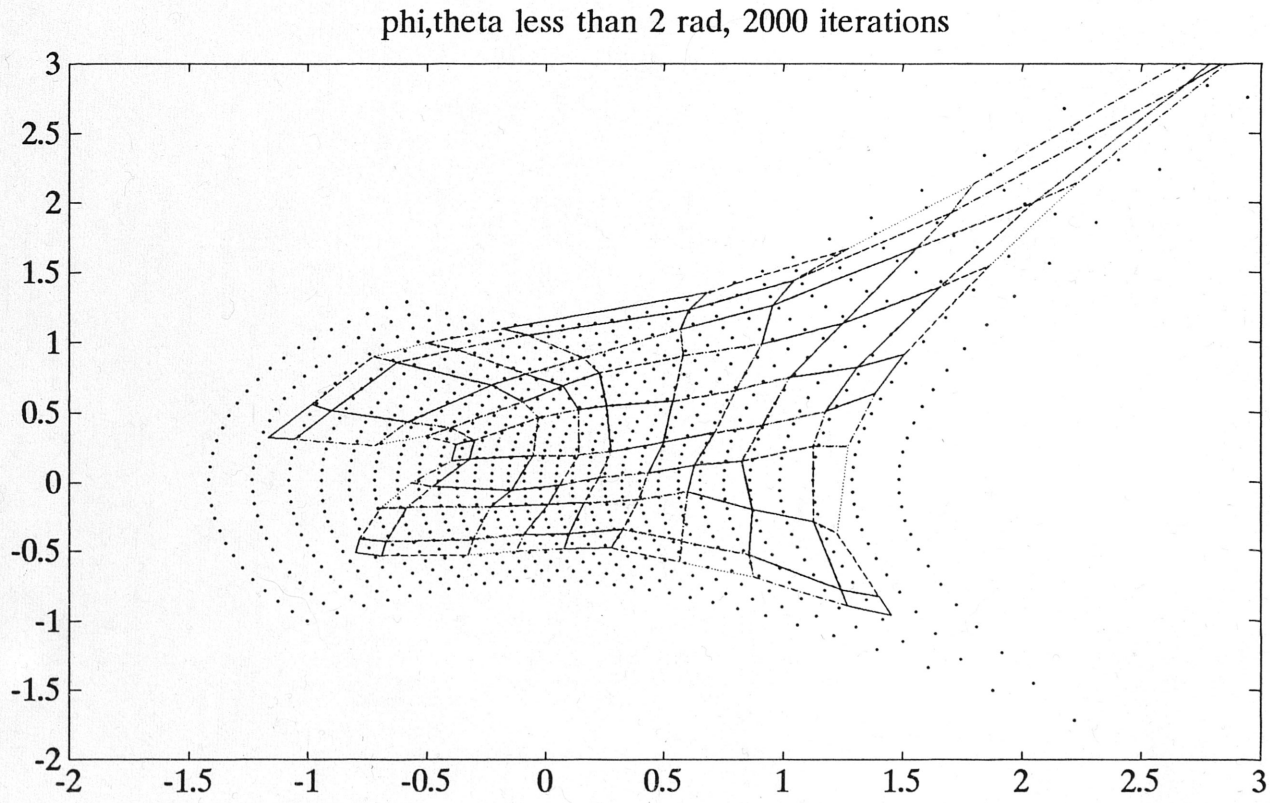


Figure (6.7) Kohonen's Algorithm : 2000 iterations (<2 rad)

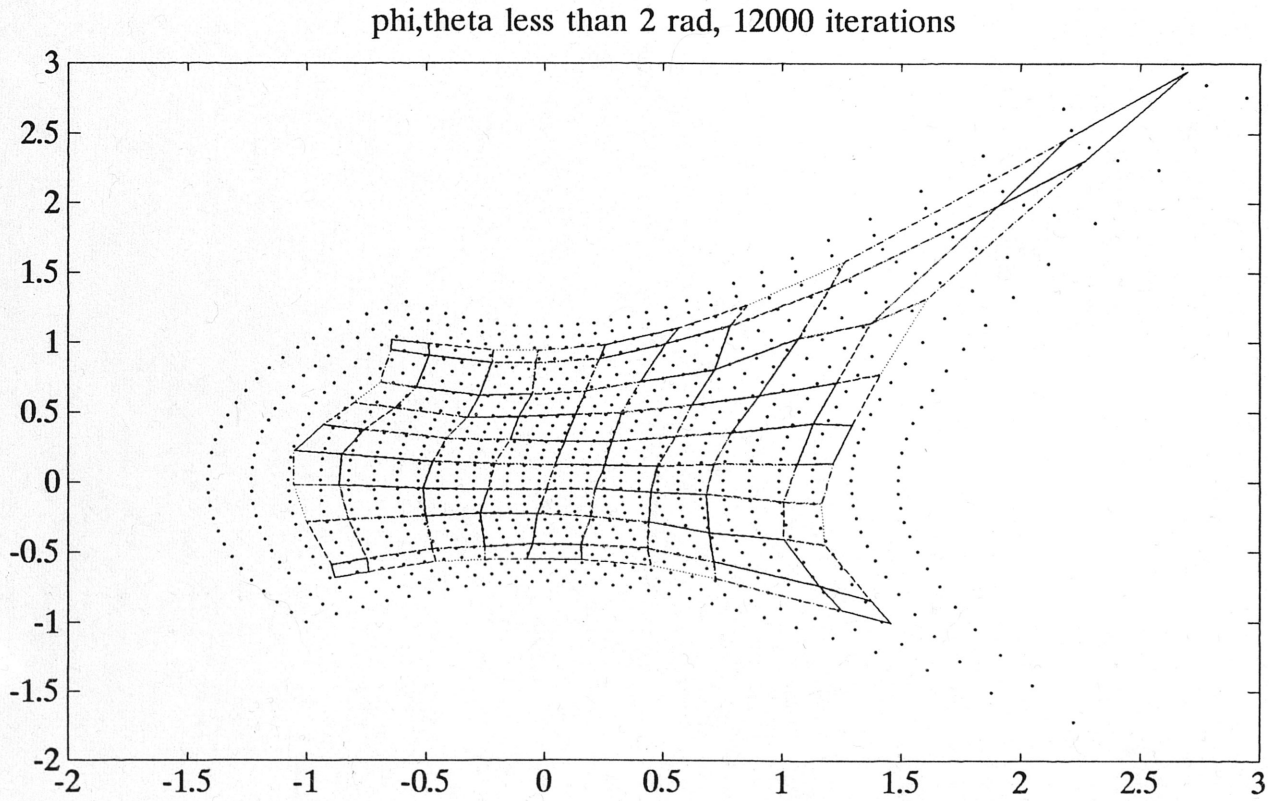


Figure (6.8) Kohonen's Algorithm : 12000 iterations (<2 rad)

(VII) RADIAL BASIS FUNCTION APPROXIMATION

With the numerous methods available for approximation, the choice of a specific algorithm clearly must be based upon the problem at hand. Once the approximation centers have been calculated using the Kohonen Topology mapping, a radial basis function method is used to approximate the desired input/output relationship. Several factors have been crucial in selecting this approach

(1) Because of the numberous data samples taken, it is not feasible to interpolate all of the data.
(2) As noted in [PG],
"From the point of view of learning as approximation, the problem of learning a smooth mapping from examples is ill-posed. . . in the sense that information in the data is not sufficient to reconstruct uniquely . The mapping in regions where data are not available."
(3) Experimental data is always a subject to noise, and radial basis function approximation can be an effective means of regularizing noisy data.
(4) It is an amazing historical fact that the many neural network algorithms have been derived from ". . . both anatomical and physiological evidence [exists] from the mammalian brains for lateral interaction . . . The degree of lateral interaction is usually described as having the form of a Mexican hat." On need compare the empirical, biological localized lateral response figure (7.1) with a typical radial basis fucntion in figure (7.2) to note the remarkable similarity in form.

The approximation of the input/output map associated with the camera variables (x,y,X,-Y,Z,ϕ,θ,ψ) using radial basis functions starts by assuming that the form of the equation by writing

$$y = F(x) = \sum_{\alpha} c_{\alpha} h(\|x - m_{\alpha}\|)$$

where h(.) can be selected from any of serveral candidate radial functions

$$h(r) = e^{-(\frac{r}{c})^2}$$

$$h(r) = (c^2 + r^2)^{-k}$$

$$h(r) = (c^2 + r^2)^{j}$$

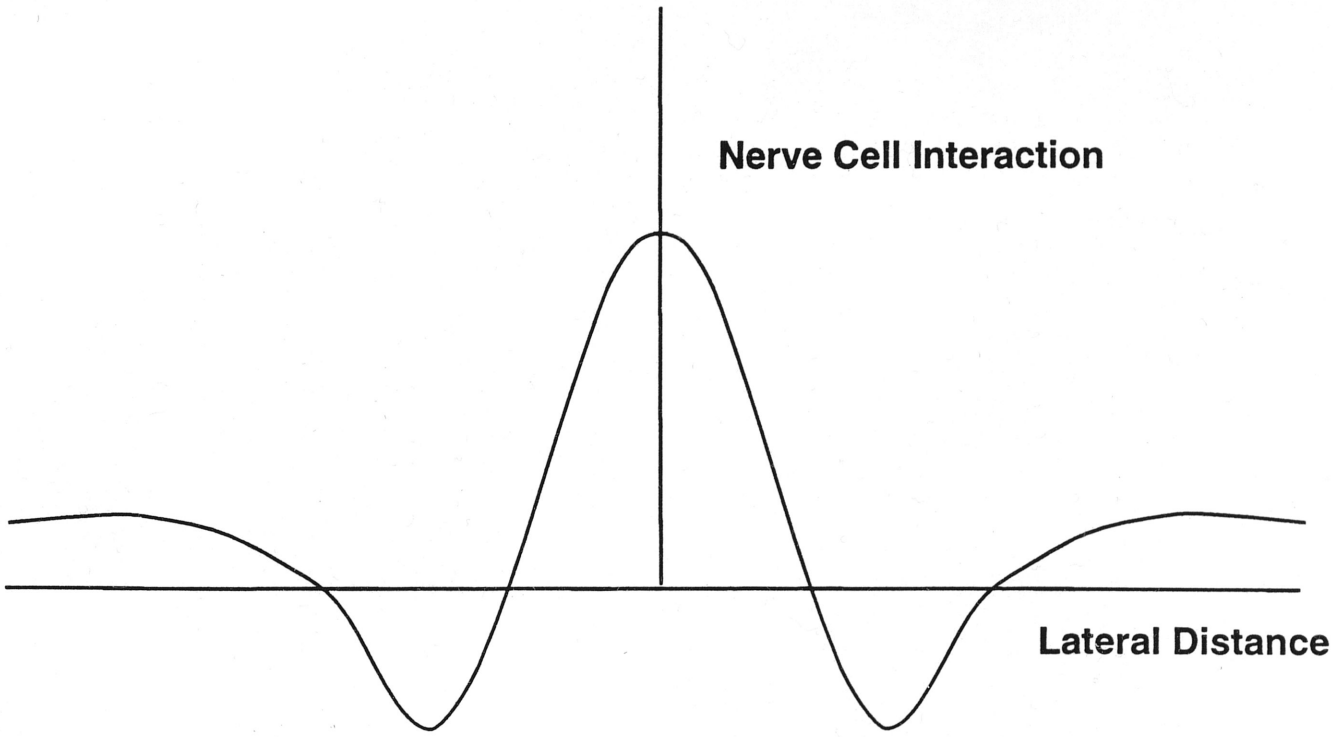


Figure (7.1) : The "Mexican Hat Function" of Lateral Interaction

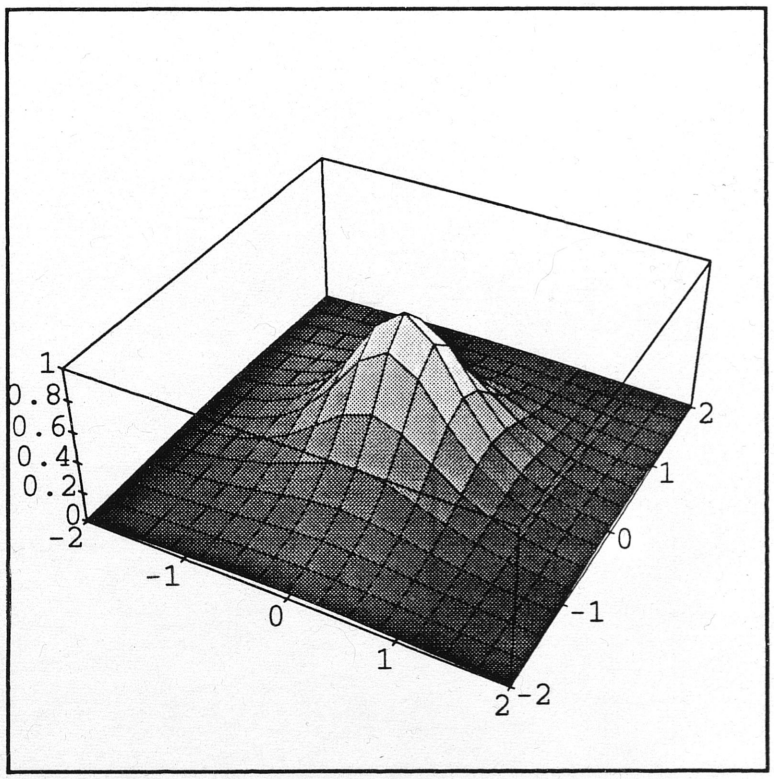


Figure (7.2) : The Radial Basis Function h(r)=(1+r*r)^(-2)

Imposing the (numerous) data pairs collected from the experiment

$$y_i = F(x_i)$$

leads to the overdetermined system of equations

$$y_i = F(x_i) = \sum_{\alpha} c_\alpha h(\|x_i - m_\alpha\|)$$

In matrix form, these equations can be written

$$y = Hc$$

where H is a S x N matrix with X>>N. The rectangular matrix

$$[H]_{i\alpha} = h(\|x_i - m_\alpha\|)$$

cannot be inverted in a classical sense, although the above system of equations can be solved via the pseudo-inverse

$$H^\dagger = (H^T H)^{-1} H^T$$
$$c = H^\dagger y$$

Numerically stable methods for calculating this least squares problem are discusseed in [Van Loan].

(VIII) EXAMPLE LEARNING SESSIONS

The overall camera orientation learning algorithm outlined in section (v),

(I) Data collection in real-time of (subsets of) data (x,y,X,Y,Z,ϕ,θ,ψ) while f, x_0, y_0, X_c Y_c, and Z_c are fixed.

(II) Neural learning to locate the centers (control points) for radial basis approximation.

(III) Solution of the radial basis approximation problem to identify the desired input/output relationship.

has been carried out for two example cases,

(i) a 2D camera orientation estimation using simulated data,
(ii) and a 1D camera orientation estimation process using real data.

In the first simulation, the focal plane coordinates (x,y) and orientation angles (Φ,Θ) have been generated using the program COGEN.C described in the appendices. During this simulation, the camera parameters

$$\begin{aligned} focal\ length &= 1. \\ principal\ offset\ (x_0, y_0) &= (0.1, 0.1) \end{aligned}$$

are fixed, as is the object space point

$$(X, Y, Z) = (1, 1, 1)$$

A disconnected grid consisting consisting of two patches of data, each consisting of 12 x 12 data points, has been generated in the focal plane. The resulting focal plane is depicted in figure (8.1). Figures (8.2) through (8.4) illustrate the 100 approximation centers generated in the focal plane to represent the 288 data points. Figures (8.5) and (8.6) depict the approximate map relating the *inverse kinematics relationships*

$$\begin{aligned} (x, y) &\rightarrow \Phi \\ (x, y) &\rightarrow \Theta \end{aligned}$$

Verification checks on the maximum error in the surfaces represented in figures (8.5) and (8.6) have been shown to be O(.01), and quite good for the simple method employed. *Perhaps the most significant attribute of the approximation method shown is its ability to provide global, smooth approximations, even in regions in which data is not available.*

The second learning session employs data directly from the experiment depicted in earlier chapters. For convenience 150 data points have been collected. All of the data points collected have the same elevation angle (=10 deg) due to the fact that only one axis is operational at this time. During the experiment, the angle Θ varies over a range of about 15 degrees as shown in figure (8.7). The centers of approximation generated by the Kohonen algorithm are also shown in figures (8.7) and (8.8). It should be noted that only the data collected at the beginning and end of the experiment have been used to generate the approximation centers, as depicted in figure (8.8). The reconstructed approximations to the *forward kinematics problem*

$$(\Phi, \Theta) \rightarrow x$$
$$(\Phi, \Theta) \rightarrow y$$

are depicted in figures (8.9) and (8.10). Again, excellent performance is achieved in the approximation in that the maximum error is of O(1/400) in this case. The control points used in the reconstruction are shown in figure (8.11). *Again, perhaps the most noteworthy attribute of the method is its ability to generate smooth data accurately away from the centers of approximation.*

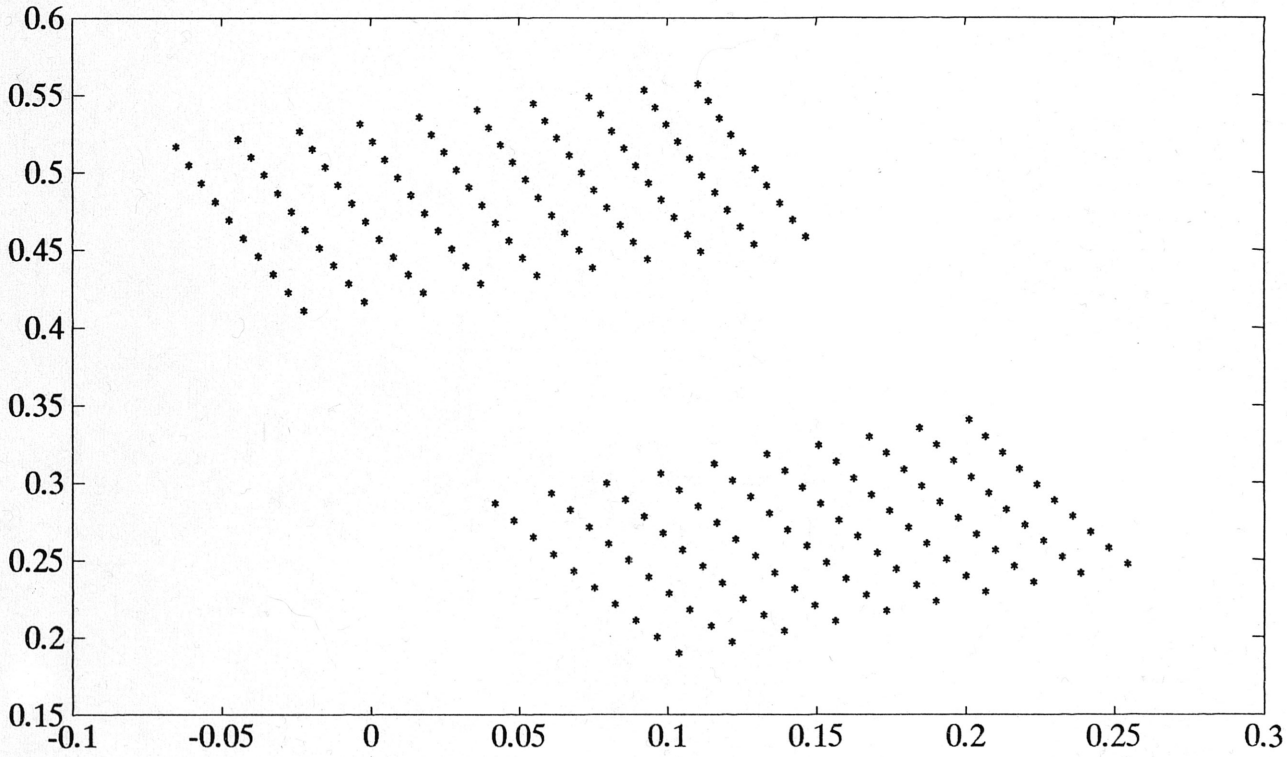


Figure (8.1) 2D Simulation Focal Plane Data Points

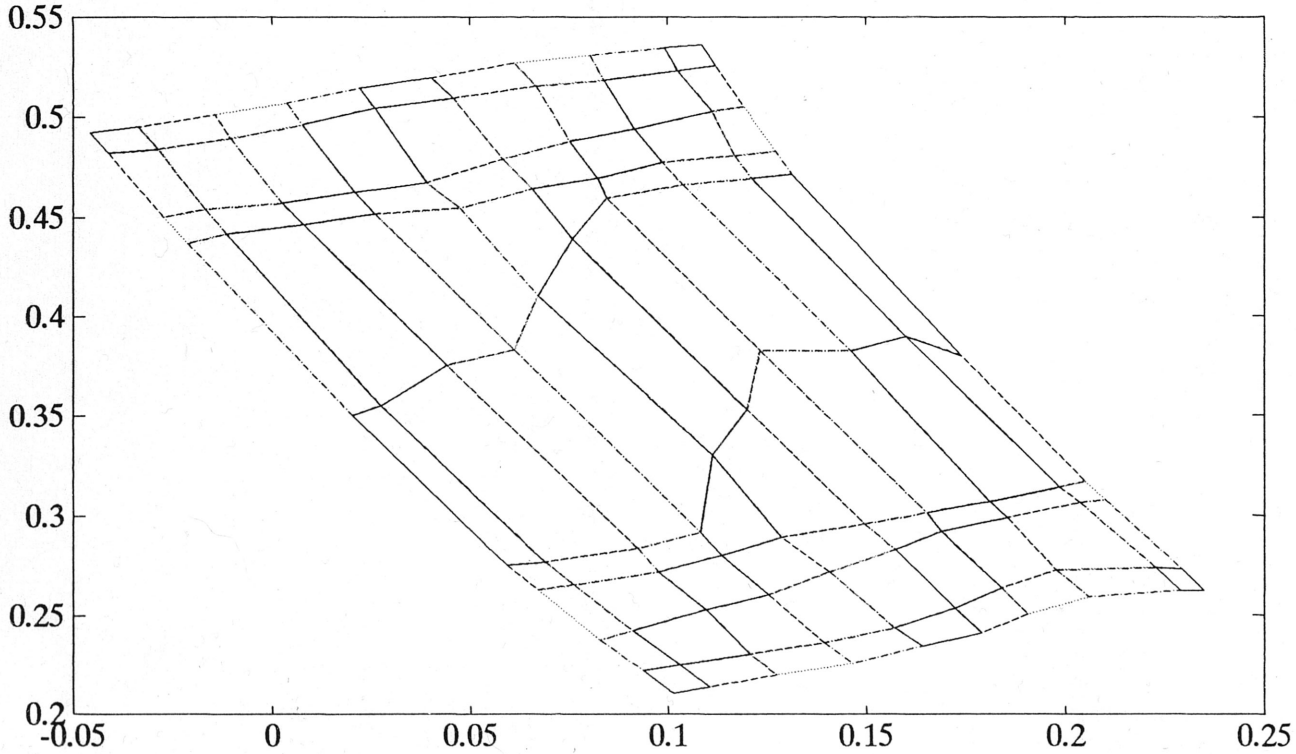


Figure (8.2) 2D Simulation Approximation Centers, 1000 iterations

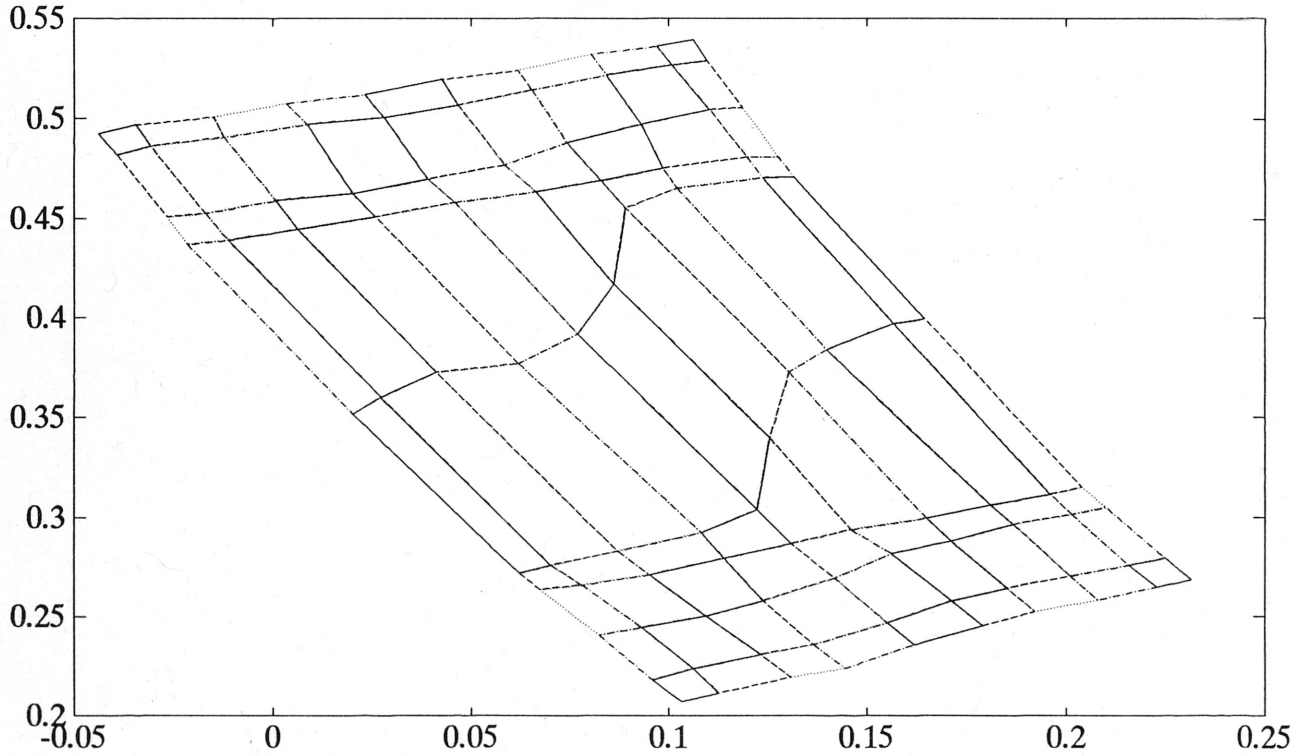


Figure (8.3) 2D Simulation Approximation Centers, 10000 iterations

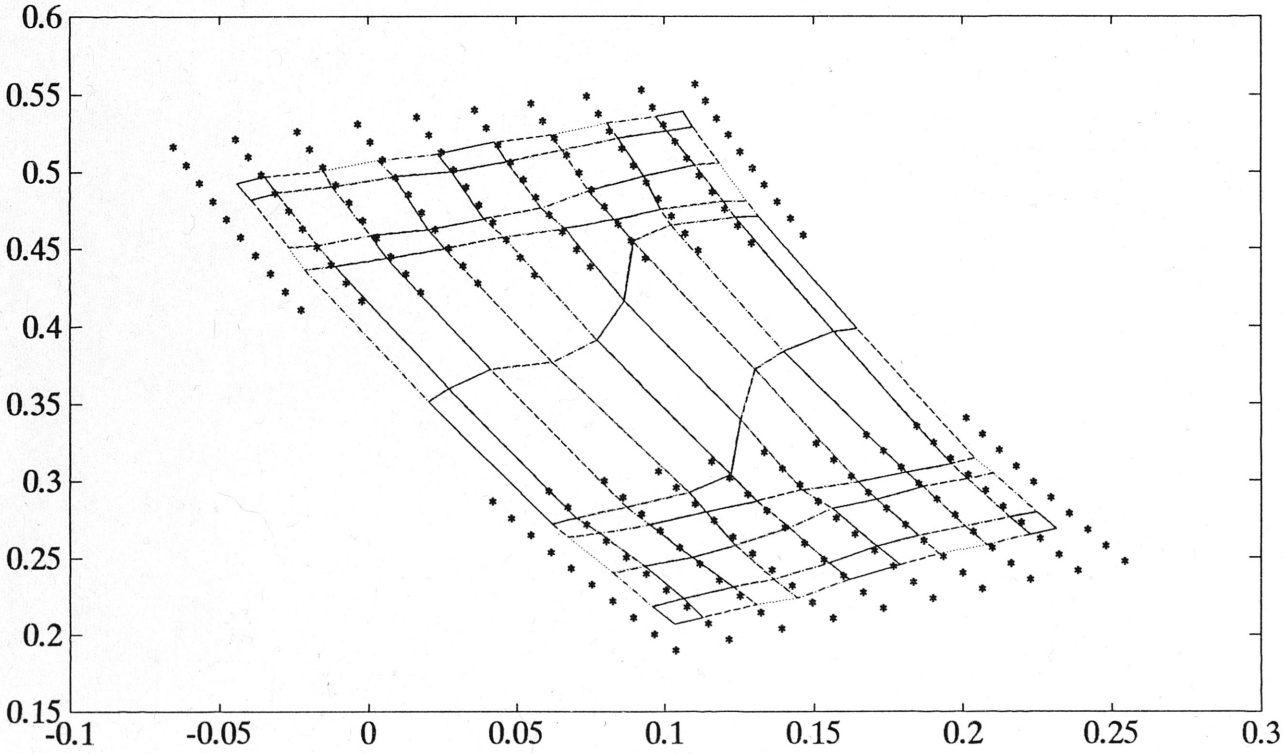


Figure (8.4) 2D Simulation Approximation Centers and Data

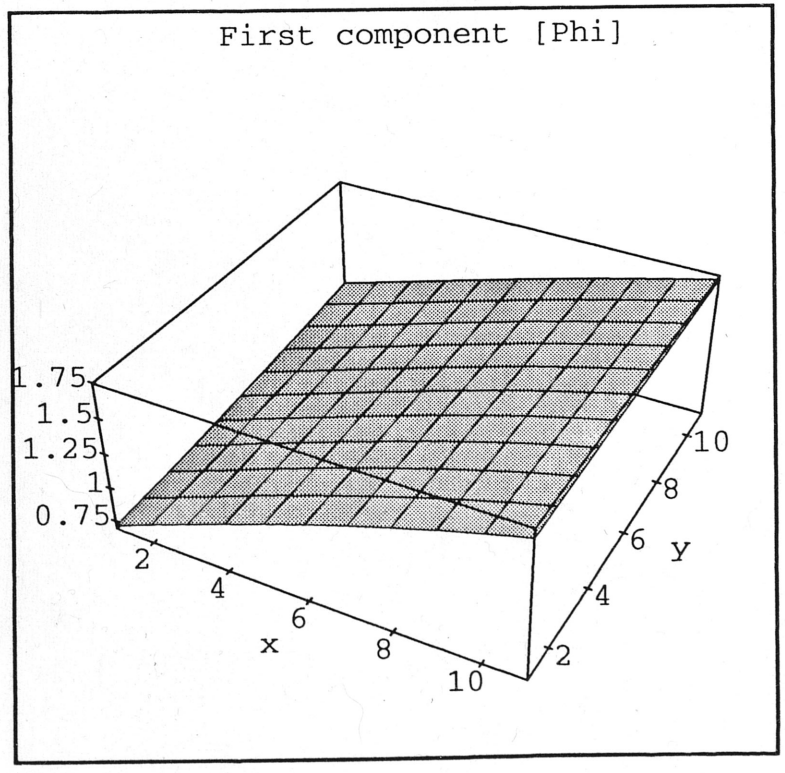


Figure (8.5) 2D Simulation, Approximation of Phi

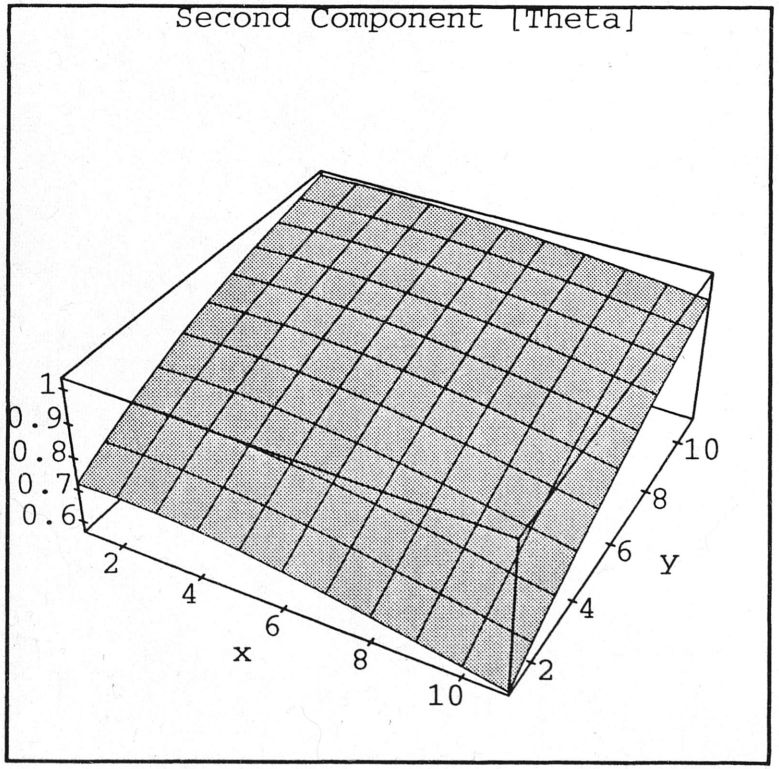


Figure (8.6) 2D Simulation, Approximation of Theta

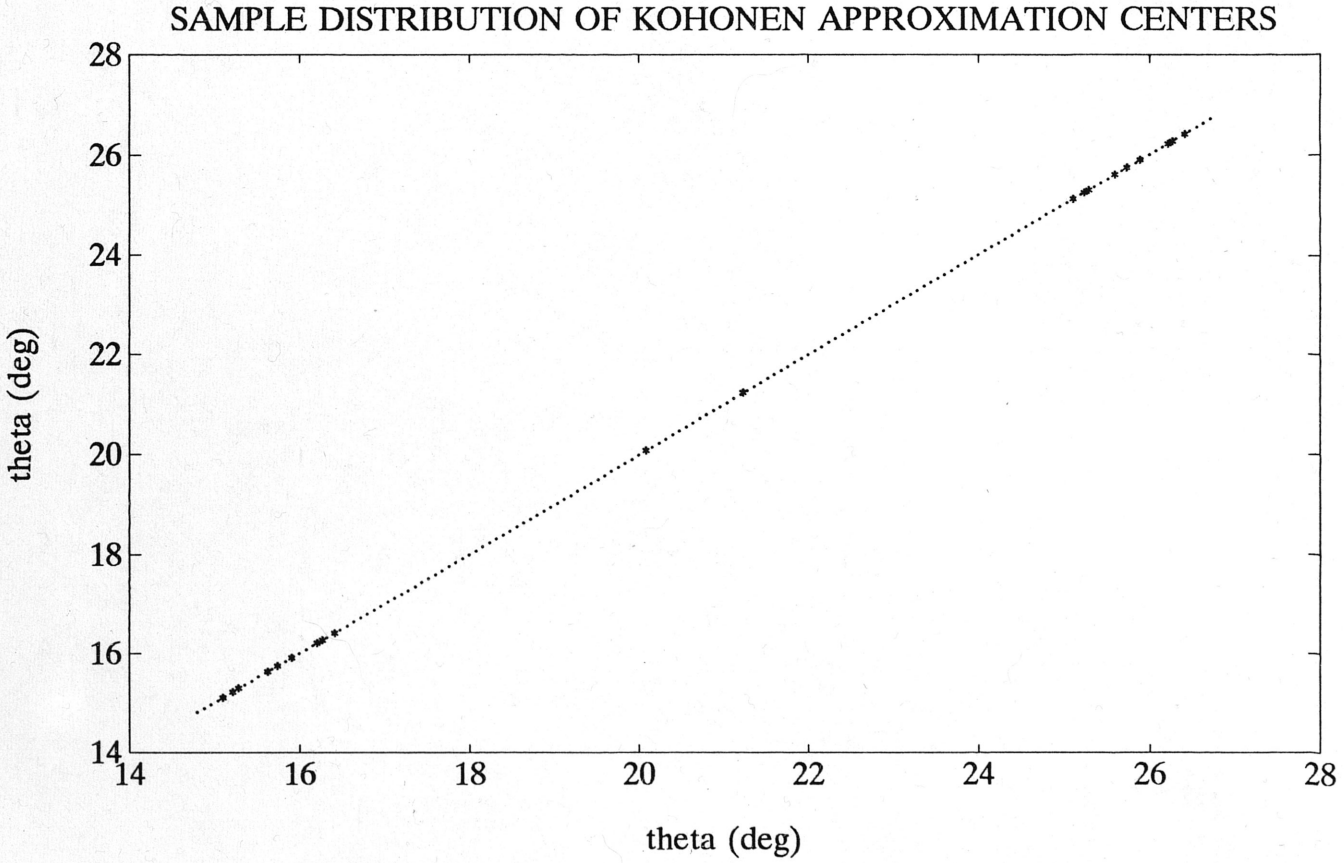


Figure (8.7) 1 Axis Experiment, Approximation Centers

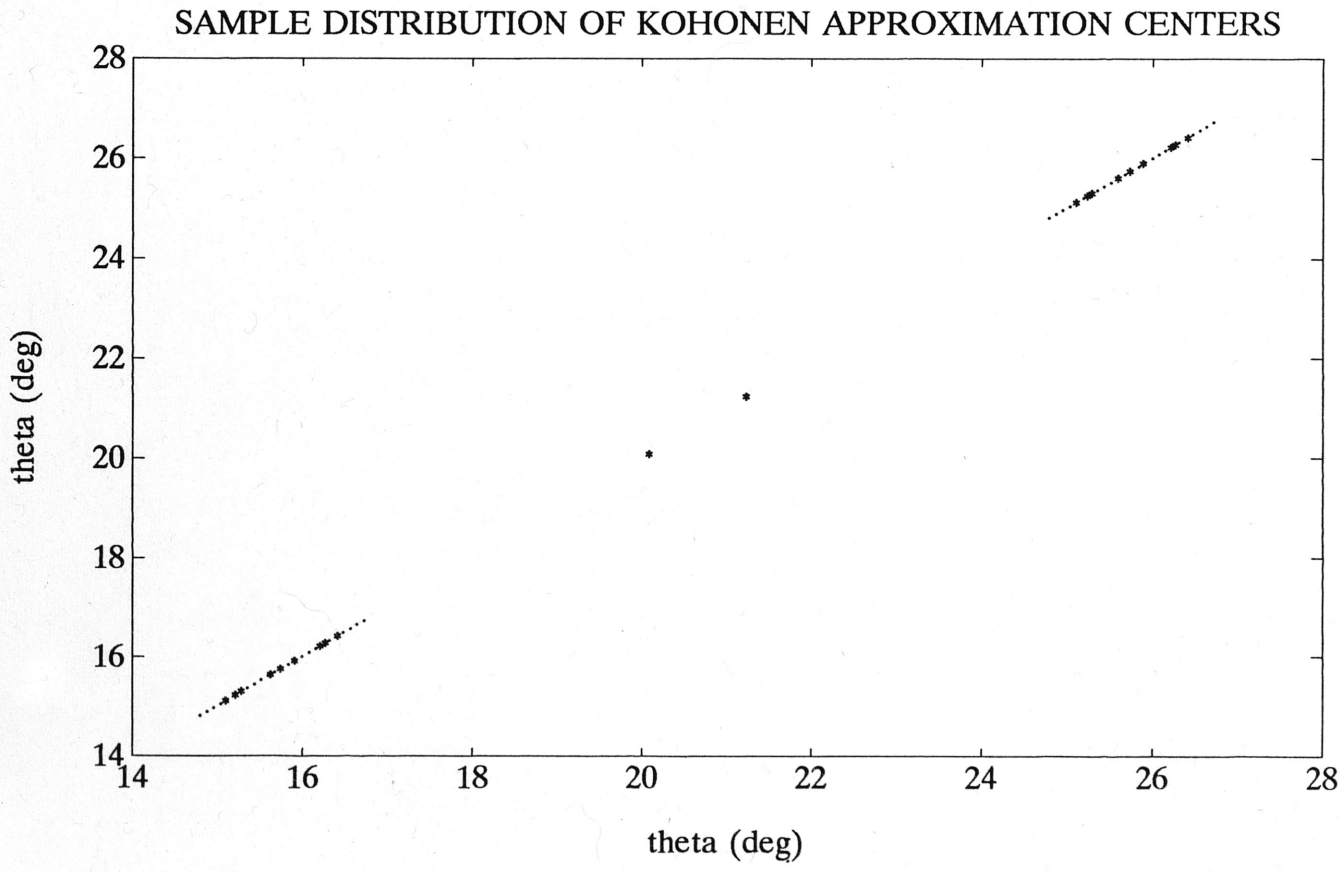


Figure (8.8) 1 Axis Experiment, Approximation Centers

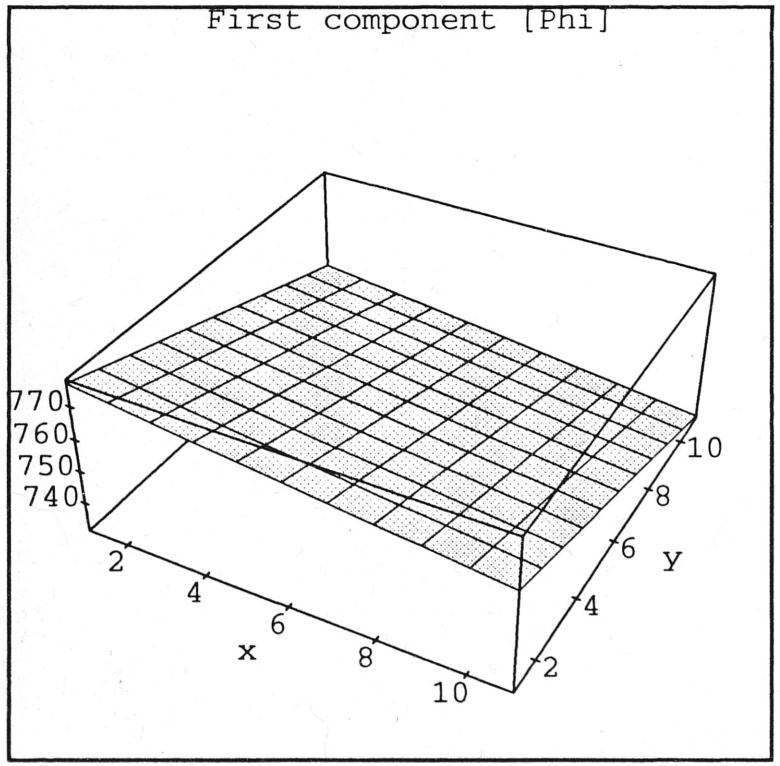


Figure (8.9) 1 Axis Experiment, Approximation of x

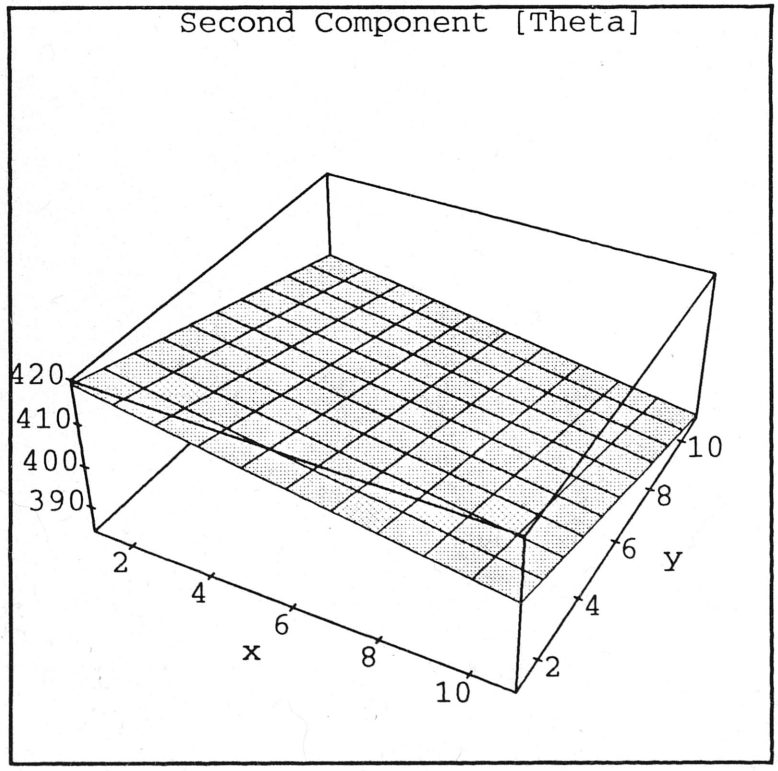


Figure (8.10) 1 Axis Experiment, Approximation of y

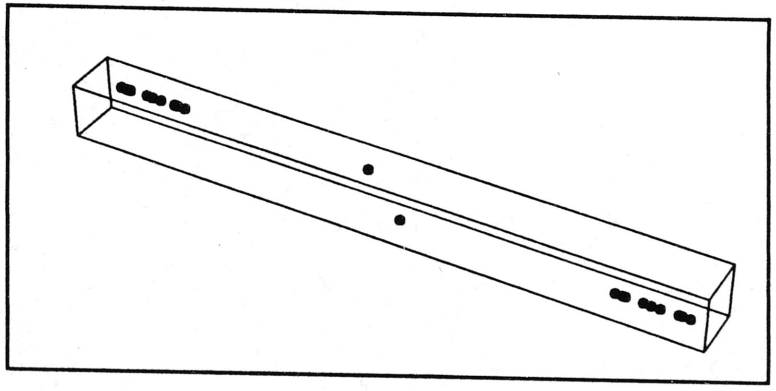


Figure (8.11) 1 Axis Experiment, Approximation Centers in 3D

(IX) CONCLUSIONS

This project has been successful in establishing a multipurpose experiment that will be the basis of ongoing research in nonlinear dynamics and control. This experiment is unique from other experimetns in the Dynamics and Control Laboratory in that it is well suited for

(i) large angle maneuvers of highly flexible structures,
(ii) the investigation of optical sensing/line-of-sight experiments typical in navigation and control application.

The first point above addresses the issue of the control of nonlinear systems, while the second opens avenues of research in proximity sensing vision processing and navigation.

The completion of the hardware/software design and implementation has made immediate impact on related research carried out by Professors Pilant, Narcowich, and Ward of the Department of Mathematics. Feasibility studies described in this report show that the use of Kohonen's technique can greatly reduce the number of approximating centers required to model the forward/backward kinematics of the system. In addition, the use of Kohonen's method and radial basis approximation has proven to be a promising method of learning the kinematics relationships. This approximation of kinematics will be vital in further research in the control of flexible space structures.

Still, a great deal of work has yet to be carried out. First, a second axis must be added to the structure before it becomes fully operational. In addition, a careful and detailed analysis of the error induced in the kinematics approximation must be carried out. Finally, since it has been the desire of the designer to enable control and vibration suppression of flexible structures, it remains to assemble the flexible "pointing superstructure and laser."

APPENDIX A

ABPOS.C

- Purpose:

  To control the Contraves-Goertz Air Bearing to a specific angular position, then place the airbearing in off mode..

```
/*************************************************************************

        ABPOS.C:  This program is used to move the airbearing to a
                  specific angular position, then place the air
                  bearing in off mode.

*************************************************************************/

#define CARD1 0x303
#define CARD2 0x353
#define PA1 0x300
#define PB1 0x301
#define PC1 0x302
#define PA2 0x350
#define PB2 0x351
#define PC2 0x352
#define CONTWORD1 0x92
#define CONTWORD2 0x88
#define STROBE ~0x1
#define RESET ~0x0
#include <stdio.h>
#include <conio.h>
#include <bios.h>

int flag,flagi;

main()
{
    char key;

/*************************************************

            Configure the cards.

*************************************************/

    outp(CARD1,CONTWORD1);
    outp(CARD2,CONTWORD2);

/*************************************************

           Initialize all ports to logic low.

*************************************************/

    outp(PC1,RESET);
    outp(PC2,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);
    flagi = inp(PC2);
    flag = flagi;

/*************************************************

           Call the following function to put
           the airbearing in position mode.

*************************************************/

    posmode();
```

```
/*************************************************

            Call function to command a position
            from the airbearing.

*************************************************/

    poscom();
    printf("Press return to put airbearing in off mode");
    scanf("%c",&key);
    offmode();

 }

/*-------------------------------------------------------*/
posmode() {

    int add,dath,datl;

    dath = 0x0;
    datl = 0x1;

/*************************************************

        Command the mode -- set address to 0101.
        See Contraves-Goertz manual, MPACS SYSTEM
        INTERFACE DEFINITION (BCD VERSION).

*************************************************/

    add = 0101;
    outp(PC1,~add);
    outp(PA2,~datl);
    outp(PB2,~dath);

/*********************************************

        Check for handshake.

*********************************************/

    while (flag == flagi) {
        flag = inp(PC2);
    }
    flag = flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/*************************************************

        Command the mode read from the mode
        registers by setting address to 005.

*************************************************/

    add = 005;
    outp(PC1,~add);
    outp(PA2,~datl);
```

```
    outp(PB2,~dath);

/*********************************************

        Check for handshake.

*********************************************/

    while (flag == flagi)  {
        flag = inp(PC2);
    }
    flag = flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/*************************************************

        Clear address output port and data
        output ports.

*************************************************/

    outp(PC1,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);

}
/*-------------------------------------------------------------*/
poscom()  {

    int add,dath,datl;

/*********************************************************

        Send the position command angle required.
        See Contraves-Goertz manual, MPACS SYSTEM
        INTERFACE DEFINITION (BCD VERSION); TR-5541B;
        page 31 for angle interpretation

        Coarse position is sent first by use of
        address 0111.

*********************************************************/

    add = 0111;
    datl = 0x11;
    dath = 0x3;

    outp(PC1,~add);
    outp(PA2,~datl);
    outp(PB2,~dath);

/*********************************************

        Check for handshake.

*********************************************/

    while (flag == flagi)  {
```

```
        flag = inp(PC2);
    }
    flag = flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/*******************************************************

        Send fine position to airbearing by use of
        address 0115.

*******************************************************/

    add = 0115;
    datl = 0x2;
    dath = 0x89;

    outp(PC1,~add);
    outp(PA2,~datl);
    outp(PB2,~dath);

/*********************************************

        Check for handshake.

*********************************************/

    while (flag == flagi)  {
        flag = inp(PC2);
    }
    flag = flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

    outp(PA2, RESET);
    outp(PB2, RESET);
    outp(PC1, RESET);
}
/*-----------------------------------------------------*/
offmode()
{
    int add,dath,datl;

/*******************************************************

        This function puts the airbearing in off mode.
        First, address the mode command.  For off mode,
        as seen on page 27, Contraves-Goertz manual:
        MPACS SYSTEM INTERFACE DEFINITION (BCD VERSION),
        TR-5541B, dath and datl need to be set to 0.

*******************************************************/

    add=0101;
    dath=0x0;
    datl=0x0;

    outp(PC1,~add);
```

```
    outp(PA2,~datl);
    outp(PB2,~dath);

    while(flag==flagi) {
        flag = inp(PC2);
    }
    flag=flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/*****************************************************

        Command the mode read, address 005.

*****************************************************/

    add = 005;
    outp(PC1,~add);
    outp(PA2,~datl);
    outp(PB2,~dath);

    while(flag==flagi) {
        flag = inp(PC2);
    }
    flag=flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/*************************************************

        Reset address and output ports.

*************************************************/

    outp(PC1,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);

}
```

ABRATE.C

- Purpose:

  To control the Contraves-Goertz Air Bearing to a specific angular rate, then place the airbearing in off mode..

```
/*****************************************************************

        ABRATE.C:  This program places the Contraves-Goertz
                   airbearing in precision rate mode, moving
                   the airbearing at a constant angular rate.

*****************************************************************/

#define CARD1 0x303
#define CARD2 0x353
#define PA1 0x300
#define PB1 0x301
#define PC1 0x302
#define PA2 0x350
#define PB2 0x351
#define PC2 0x352
#define CONTWORD1 0x92
#define CONTWORD2 0x88
#define STROBE ~0x1
#define RESET ~0x0
#include <stdio.h>
#include <conio.h>
#include <bios.h>

int flag,flagi;

main()
{
    char key;
    int fdatl, fdath, cdatl, cdath;

/************************************************

        Configure the cards.

************************************************/

    outp(CARD1,CONTWORD1);
    outp(CARD2,CONTWORD2);

/************************************************

        Initialize all ports to logic low.

************************************************/

    outp(PC1,RESET);
    outp(PC2,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);
    flagi = inp(PC2);
    flag = flagi;

/************************************************

        The following function commands the
        airbearing to the precision rate mode.

************************************************/
```

```
    ratemode();

/************************************************

        The following function commands the
        airbearing to execute precision rate
        (degrees/sec or Earth Rate Units, and
        CCW or CW direction depending on
        specifications in the cdath).

************************************************/

    fdatl= 0x0;
    fdath= 0x0;
    cdatl= 0x10;
    cdath= 0x0;
    prerate(fdatl,fdath,cdatl,cdath);

/************************************************

        The following will "turn off" the
        precision rate procession if any
        key is entered.

************************************************/

    printf("Press return to stop air bearing");
    scanf("%c", &key);
    fdatl=0x0;
    fdath=0x0;
    cdatl=0x0;
    cdath=0x0;
    prerate(fdatl, fdath, cdatl, cdath);
    offmode();
}
/*-------------------------------------------------*/

ratemode()
{
    int add,dath,datl;

/************************************************

        Command the Mode--octal address 0101.

************************************************/

    add = 0101;
    dath = 0x0;
    datl = 0x2;
    outp(PC1,~add);
    outp(PA2,~datl);
    outp(PB2,~dath);

/************************************************

        Check for handshake.

************************************************/
```

```
    while (flag == flagi) {
        flag = inp(PC2);
    }
    flag = flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/***********************************************

        Command mode read--octal address 005.
        The precision rate mode is called by
        datl, addressed to port A of card 2.

***********************************************/

    add = 005;
    outp(PC1,~add);
    outp(PA2,~datl);
    outp(PB2,~dath);

/***********************************************

        Check for handshake.

***********************************************/

    while(flag == flagi)  {
        flag = inp(PC2);
    }
    flag = flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

    outp(PC1,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);
}
/*-------------------------------------------------*/
prerate(fdatl,fdath,cdatl,cdath)

int fdatl, fdath, cdatl, cdath;

{
    int add;

/*********************************************************

        Send coarse read word to air bearing.
        See page 32, MPACS SYSTEM INTERFACE DEFINITION
        (BCD VERSION) for the bit pattern.

*********************************************************/

    add = 0121;
    outp(PC1,~add);
    outp(PA2,~cdatl);
    outp(PB2,~cdath);
```

```
/*******************************************

        Check for handshake.

*******************************************/

    while (flag == flagi)  {
        flag = inp(PC2);
    }
    flag = flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/*******************************************

        Send fine rate word to airbearing.

*******************************************/

    add = 0125;

    outp(PC1,~add);
    outp(PA2,~fdatl);
    outp(PB2,~fdath);

/*******************************************

        Check for handshake.

*******************************************/

    while (flag == flagi)  {
        flag = inp(PC2);
    }
    flag = flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/*******************************************

        Clear address and data output ports.

*******************************************/

    outp(PC1,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);

}
/*-----------------------------------------------*/
offmode()
{
    int add,dath,datl;

/*******************************************

        Command the mode.
```

```
*******************************************/

    add=0101;
    dath=0x0;
    datl=0x0;

    outp(PC1,~add);
    outp(PA2,~datl);
    outp(PB2,~dath);

/*******************************************

        Check for handshake.

*******************************************/

    while (flag == flagi) {
        flag=inp(PC2);
    }
    flag=flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

/*********************************************************

        Command mode read, selecting off mode  for datl.

*********************************************************/

    add = 005;
    outp(PC1,~add);
    outp(PA2,~datl);
    outp(PB2,~dath);

/*************************************

        Check for handshake.

*************************************/

    while(flag == flagi) {
        flag=inp(PC2);
    }
    flag=flagi;

    outp(PC2,STROBE);
    outp(PC2,RESET);

    outp(PC1,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);

}
```

FRAME.C

- Purpose:

  To retrieve camera data with DT2861 Frame Grabber using DTIRIS Subroutine Library.

```
/*****************************************************************

        FRAME.C:  This program illustrates taking camera data
                  with DT2861 Frame Grabber using DTIRIS
                  subroutines.

*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <iserrs.h>
#include <isdefs.h>

main()
{
        FILE *fp;
        int i, ierr, idisp, row_start, column_start, M, N;
        int  frame_number, frame_count, sync_state, ilut_number;
        int olut_number;


/*************************************************

        Initialize Frame Grabber.

*************************************************/

        printf("\n\nInitializing DT2861 Frame Grabber  ");

        ierr = IS_INITIALIZE();

        frame_number = 0;
        frame_count = 1;
        sync_state = 1;
        printf("\n  Enter starting row and column ");
        scanf("%d, %d",&row_start,&column_start);
        printf("  Enter height and width ");
        scanf("%d, %d",&M,&N);
        printf("  Enter number for input look up table ");
        scanf("%d",&ilut_number);
        printf("  Enter number for ouput look up table ");
        scanf("%d",&olut_number);
        printf("  Is display off (0) or on (1) ? ");
        scanf("%d",&idisp);
        printf("\n\n  Loading Look up tables ");

/*************************************************

        Setting parameters.

*************************************************/

        ierr = IS_SELECT_ILUT(ilut_number);
        ierr = IS_SELECT_OLUT(olut_number);
        printf("\n  Setting parameters ");
        ierr = IS_SET_SYNC_SOURCE(sync_state);
        ierr = IS_SELECT_INPUT_FRAME(0);
        ierr = IS_SELECT_OUTPUT_FRAME(0);
        ierr = IS_DISPLAY(0);
```

```
        if(idisp==1)
        {
                ierr = IS_DISPLAY(1);
        }

        ierr = IS_SET_ACTIVE_REGION(row_start,column_start,M,N);
        ierr = IS_FRAME_CLEAR(0);

        printf("\nInitialization complete\n\nAcquiring Data ");

/*************************************************

        Taking data.

*************************************************/

        ierr = IS_ACQUIRE(frame_number,frame_count);
        ierr = IS_WAIT_ACQUIRE_COMPLETE();

        printf(". . . complete ");

/*************************************************

        Turning off DT2861.

*************************************************/

        printf("\n\nTurning off Frame Grabber ");

        ierr = IS_END();

}
```

APPIXEL.C

- Purpose:

  Use the DT7020 Array Processor to determine the centroid of a dummy array formatted similiar to input from the DT2861 Frame Grabber.

```
/*************************************************************************

   APPIXEL.C:  This program uses the DT7020 array processor to determine
               the centroid of a dummy array formatted similiar to input
               from DT2861 frame grabber.

*************************************************************************/

#define NL  6L
#define ML  5L
#define N   6
#define M   5
#define NML 30L
#define NM  30

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <bios.h>
#include <math.h>
#include <isdefs.h>

main()
{
        FILE *fp;
        int i, j, k, ierr, idisp, sync_state, row_start, column_start;
        int frame_number, frame_count, ilut_number, olut_number;
        float xc,yc,isum,x,y,nm,vec[NM];
        long ipaloc(), handle1, ap_nm, ap_isum, ap_xc, ap_yc;
        long ap_in, ap_lvec, ap_kvec;
        char key;

        fp=fopen("appixel.dat","w");

/***********************************************

        Configuring array processor.

***********************************************/

        printf("\n\nConfiguring DT7020 Floating Point Array Processor");
        apstrt();
        printf(". . . continuing");

/***********************************************

        Allocate space aboard array processor.

***********************************************/

        printf("\n  Allocating space aboard array processor ");
        ap_nm = ipaloc(1L);
        ap_isum = ipaloc(1L);
        ap_xc = ipaloc(1L);
        ap_yc = ipaloc(1L);
        ap_in = ipaloc(NML);
        ap_lvec = ipaloc(NML);
        ap_kvec = ipaloc(NML);

/***********************************************
```

```
        Transfer data to array processor memory.

************************************************/

        printf("\n  Loading constant values in array processor ");
        apputv((float)NM,ap_nm);
        apputv((0.0),ap_xc);
        apputv((0.0),ap_yc);
        apputv((0.0),ap_isum);

/************************************************

        Build vectors for centroiding.

************************************************/

        printf("\n  Loading centroiding data into array processor");
        k=0;
        for(i=0; i<M; i++)
        {
                for(j=0; j<N; j++)
                {
                        vec[k] = (float) j;
                        k=k+1;
                }
        }
        apputa(vec,ap_lvec,NML);
        k=0;
        for(i=0;i<M;i++)
        {
                for(j=0;j<N;j++)
                {
                        vec[k] = (float)i;
                        k = k+1;
                }
        }
        apputa(vec,ap_kvec,NML);

/************************************************

        "vec[30]" illustrates frame returned from
        Frame Grabber filled with various gray
        scale values.

************************************************/

        for(i=0;i<7;i++)
        {
                vec[i] = 0.0;
        }
        vec[7] = 36.0;
        for(i=8;i<12;i++)
        {
                vec[i] = 0.0;
        }
        vec[12] = 45.0;
        vec[13] = 50.0;
        vec[14] = 28.0;
        for(i=15;i<19;i++)
```

```
        {
                vec[i] = 0.0;
        }
        vec[19] = 40.0;
        for(i=20;i<30;i++)
        {
                vec[i] = 0.0;
        }

/************************************************

        Load frame to array processor.

************************************************/

        printf("\n  Loading frame to array processor ");
        apputa(vec,ap_in,NML);

/************************************************

        Build macros.

************************************************/

        printf("\n  Building array processor macros ");
        apmac(&handle1);
                rdot(ap_in,ap_lvec,ap_xc,ap_nm);
                rdot(ap_in,ap_kvec,ap_yc,ap_nm);
                rsum(ap_in,ap_isum,ap_nm);
        apendm(handle1, 0L);


/************************************************

        Initialization complete.
        Finding centroid.

************************************************/

        printf("\n  Finding centroid ");
        printf("\n     Waiting . . .");
        apgo(handle1);
        apwait();
        printf("Continuing ");

/************************************************

        Retrieve values from array processor.

************************************************/

        printf("\n  Retrieving data from array processor ");
        apgeta(ap_xc,&xc,1L);
        apgeta(ap_yc,&yc,1L);
        apgeta(ap_isum,&isum,1L);
        column_start = 0.0;
        row_start = 0.0;

        x = (float) column_start + xc/isum;
        y = (float) row_start + yc/isum;
```

```
/************************************************

        Printing values to data file.

************************************************/

        printf("\n  x = %f  y = %f ",x,y);
        printf("\n  Printing data to file ");
        fprintf(fp," x = %f   y = %f ",x,y);

/************************************************

        Turn off equipment.

************************************************/

        printf("\n\nTurning off Array Processor\n ");
        apstop();
}
```

UNIFORM.C

- Purpose:

  Move the Contraves-Goertz Air Bearing through a series of angular positions at 0.8°increments. At each location, the center pixel location for the reference point is determined.

- OUTPUT

  (θ, φ) and corresponding reference point center pixel location

```
/*************************************************************************

    UNIFORM.C:  This program moves the airbearing at 0.08 degree increments
                beginning at 14.8 degrees angular position. Number of data
                points required is a varible input from screen.  The centroid
                of the same fixed point is found for each angular position,
                then written to a data file.

*************************************************************************/

#define CARD1  0x303
#define CARD2  0x353
#define PA1  0x300
#define PB1  0x301
#define PC1  0x302
#define PA2  0x350
#define PB2  0x351
#define PC2  0x352
#define CONTWORD1 0x92
#define CONTWORD2 0x88
#define STROBE ~0x1
#define RESET ~0x0
#define NL 48L
#define ML 60L
#define N 48
#define M 60
#define NML 2880L
#define NM  2880


#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <bios.h>
#include <math.h>
#include <isdefs.h>
#include <iserrs.h>

int flag,flagi;

main()
{
        FILE *fp;
        int i, j, k, num,ierr, idisp, row_start, column_start;
        int frame_number, frame_count, sync_state, ilut_number;
        int olut_number;
        float theta1, theta2, dtheta, xc, yc, isum, x, y, vec[NM];
        long ipaloc(), fill, handle1, ap_nm, ap_isum, ap_xc, ap_yc;
        long ap_in, ap_lvec, ap_kvec;
        char key;

        fp = fopen("uniform.dat","w");

/************************************************

        Initializing air bearing.

************************************************/

        printf("Configuring Contraves-Goertz Airbearing . . .");
```

```
        outp(CARD1,CONTWORD1);
        outp(CARD2,CONTWORD2);

/************************************************

        Initializing all ports to logic low.

************************************************/

        outp(PC1,RESET);
        outp(PC2,RESET);
        outp(PA2,RESET);
        outp(PB2,RESET);

        flagi = inp(PC2);
        flag = flagi;

/************************************************

        Initializing array processor.

************************************************/


        printf(" done. \n\nConfiguring DT7020 Array Processor . . . ");
        apstrt();

/************************************************

        Allocate space on array processor board.

************************************************/

        ap_nm = ipaloc(1L);
        ap_isum = ipaloc(1L);
        ap_xc = ipaloc(1L);
        ap_yc = ipaloc(1L);
        ap_in = ipaloc(NML);
        ap_lvec = ipaloc(NML);
        ap_kvec = ipaloc(NML);

/************************************************

        Transfer data to array processor memory.

************************************************/

        apputv((float)NM,ap_nm);
        apputv((0.0),ap_xc);
        apputv((0.0),ap_yc);
        apputv((0.0),ap_isum);

/************************************************

        Build vectors for centroiding.

************************************************/

        k=0;
```

```
        for(i=0;i<M;i++)
        {
                for(j=0;j<N;j++)
                {
                        vec[k] = (float) j;
                        k=k+1;
                }
        }
        apputa(vec,ap_lvec,NML);
        k=0;
        for(i=0;i<M;i++)
        {
                for(j=0;j<N;j++)
                {
                        vec[k] = (float) i;
                        k=k+1;
                }
        }
        apputa(vec,ap_kvec,NML);

/************************************************

        Building Macros for array processor.

************************************************/

        apmac(&fill);
                rgetb(ap_in,ap_nm);
        apendm(fill,0L);

        apmac(&handle1);
                rdot(ap_in,ap_lvec,ap_xc,ap_nm);
                rdot(ap_in,ap_kvec,ap_yc,ap_nm);
                rsum(ap_in,ap_isum,ap_nm);
        apendm(handle1,0L);

        printf(". . . done");

/************************************************

        Initializing Frame Grabber.

************************************************/

        printf("\nConfiguring DT2861 Frame Grabber . . . ");

        frame_number = 0;
        frame_count = 1;
        sync_state = 1;
        ilut_number = 0;
        olut_number = 0;
        printf("\nIs display off (0) or on (1) ? ");
        scanf("%d",&idisp);

        ierr = IS_INITIALIZE();
        ierr = IS_SET_SYNC_SOURCE(sync_state);
        ierr = IS_SELECT_ILUT(ilut_number);
        ierr = IS_SELECT_OLUT(olut_number);
        ierr = IS_SELECT_INPUT_FRAME(0);
        ierr = IS_SELECT_OUTPUT_FRAME(0);
```

```
        ierr = IS_DISPLAY(0);
        if(idisp==1)
        {
                ierr = IS_DISPLAY(1);
        }

/************************************************

        All initialization complete.
        Beginning Data acquisition.

************************************************/

        printf("\n\nAll initialization is complete.");
        printf("\n\n Enter number of data points. ");
        scanf("%d",&num);
        printf("\n\nEnsure Airbearing is in remote mode, then press enter.");
        scanf("%c",&key);

/************************************************

         Define theta2 as constant due to only
         one degree of freedom, theta1.

         Data is taken with a uniform slew manuever.

************************************************/

        theta2 = 10.0;
        row_start = 0;
        column_start = 208;
        dtheta = 0.0800;
        theta1 = 14.8000;
        for(i=0;i<num;i++)
        {

/************************************************

        Positioning the air bearing.

************************************************/

                printf("\nBegining position manuever, %6.4f degrees . .",theta1)
                AB_posmode(theta1);

/************************************************

       Acquiring data.

************************************************/

                printf("\n . . . . acquiring data . . . . ");

                ierr = IS_FRAME_CLEAR(0);
                ierr = IS_SET_ACTIVE_REGION(row_start,column_start,M,N);
                ierr = IS_ACQUIRE(frame_number,frame_count);
                ierr = IS_WAIT_ACQUIRE_COMPLETE();
                ierr = IS_WRITE_EPORT();
                apgo(fill);
                ierr = IS_WAIT_EPORT_COMPLETE();
```

```
                apwait();

/************************************************

        Finding Centroid of frame (center pixel
        location of reference point).

************************************************/

                apgo(handle1);
                apwait();
                apgeta(ap_xc,&xc,1L);
                apgeta(ap_yc,&yc,1L);
                apgeta(ap_isum,&isum,1L);

                x = (float) column_start + xc/isum;
                y = (float) row_start + yc/isum;

/************************************************

        Printing corresponding locations to
        data file.

************************************************/

                fprintf(fp,"\n %8.4f  %8.4f  %6.2f  %6.2f ",theta1,theta2,x,y);

                theta1 = theta1 + dtheta;
                row_start = row_start + 2;
        }


        fclose(fp);

/************************************************

        Shut down procedure.

************************************************/

        printf("\n\nBeginning shut down...Airbearing...");
        AB_offmode();
        printf("Array Processor...");
        apstop();
        printf("Frame Grabber.");
        ierr = IS_END();
}

/*****************************************************************
:
:        Contraves-Goertz Airbearing Position Subroutines
:
*****************************************************************/
AB_posmode (theta1)

float theta1;

{

/* AB_posmode places the airbearing in position mode.  This subroutine
```

```
also separates the desired angular position, theta1, for position manuever
in AB_poscom.    */

        int address, dath, datl;
        int a,b,c,d,e,f,g;
        float ntheta1;

/* Commanding the mode -- octal address 0101.  */

        address = 0101;
        datl = 0x1;
        dath = 0x0;

        outp(PC1,~address);
        outp(PA2,~datl);
        outp(PB2,~dath);

/* Check for handshaking. */

        while(flag==flagi)
        {
                flag = inp(PC2);
        }
        flag=flagi;

        outp(PC2,STROBE);
        outp(PC2,RESET);

/* Command mode read -- octal address 005.  */

        address = 005;

        outp(PC1,~address);
        outp(PA2,~datl);
        outp(PB2,~dath);

/* Check for handshaking. */

        while(flag==flagi)
        {
                flag = inp(PC2);
        }
        flag = flagi;

        outp(PC2,STROBE);
        outp(PC2,RESET);

/* Reset data output ports. */

        outp(PA2,RESET);
        outp(PB2,RESET);
        outp(PC1,RESET);

/* Separate theta1 to command the position.  */

        a = (int) (theta1/100.);
        ntheta1 = theta1 - a*100.;
        b = (int) (ntheta1/10.);
        ntheta1 = ntheta1 - b*10.;
        c = (int) (ntheta1/1.0);
```

```
        ntheta1 = ntheta1 - c;
        d = (int) (ntheta1/0.1);
        ntheta1 = ntheta1 - d*0.1;
        e = (int) (ntheta1/0.01);
        ntheta1 = ntheta1 - e*0.01;
        f = (int) (ntheta1/0.001);
        ntheta1 = ntheta1 - f*0.001;
        g = (int) (ntheta1/0.0001);

/* Command the position. */

        AB_poscom(a,b,c,d,e,f,g);
}

/*-------------------------------------------------------------*/

AB_poscom(a,b,c,d,e,f,g)

int a, b, c, d, e, f, g;

{

/* AB_poscom sends the desired angular position to airbearing. */

        int address,fdatl,fdath,cdatl,cdath;

/* Send coarse position -- octal address 0111. */

        address = 0111;
        cdath = a*16 + b;
        cdatl = c*16 +d;

        outp (PC1,~address);
        outp (PA2,~cdatl);
        outp (PB2,~cdath);

/* Check for handshake. */

        while (flag == flagi)
        {
              flag = inp(PC2);
        }
        flag = flagi;

        outp(PC2,STROBE);
        outp(PC2,RESET);

/* Send fine position -- octal address 0115.  */

        address = 0115;
        fdath = e ;
        fdatl = f*16 + g;

        outp(PC1,~address);
        outp(PA2,~fdatl);
        outp(PB2,~fdath);

/* Check for handshake.  */

        while(flag==flagi)
```

```
        {
                flag = inp(PC2);
        }
        flag = flagi;

        outp(PC2,STROBE);
        outp(PC2,RESET);

        outp(PC1,RESET);
        outp(PA2,RESET);
        outp(PB2,RESET);

/* Allow for completion of manuever. */

        AB_posread(a,b,c,d,e,f,g);

}

/*-------------------------------------------------------------*/
AB_posread(a,b,c,d,e,f,g)

/* This function will keep reading and displaying the position until the
   air bearing attains the commanded attitude. */

int a,b,c,d,e,f,g;
{

int d1,d2,d3,d4,d5,d6,d7,addc,addf;
int dath,datl,num,IFLAG=1,index=0;
        addc=011;
        addf=015;
        num=0xff00;
        while (IFLAG)
        {
                switch (index)
                {
                        case 0:
                                if ((a-d1) == 0)
                                index=1; break;
                        case 1:
                                if ((b-d2) == 0)
                                index=2; break;
                        case 2:
                                if ((c-d3) == 0)
                                index=3; break;
                        case 3:
                                if ((d-d4) == 0)
                                index=4; break;
                        case 4:
                                if ((e-d5) == 0)
                                index=5; break;
                        case 5:
                                if ((f-d6) == 0)
                                index=6; break;
                        case 6:
                                if (((g-d7) == 0) && ((f-d6) == 0))
                                IFLAG=0; break;
                }
                if (IFLAG == 0)
                {
```

```
                        goto PQ;
                }
                outp(PC1,~addc);
                while (flag==flagi)
                {
                        flag=inp(PC2);
                }
                flag=flagi;
                dath= (~inp(PB1));
                datl= (~inp(PA1));
                dath -= num;
                datl -= num;
                d1=dath/16;
                d2=dath-d1*16;
                d3=datl/16;
                d4=datl-d3*16;
                outp(PC1,~addf);
                while (flag==flagi)
                {
                        flag=inp(PC2);
                }
                flag=flagi;
                dath= (~inp(PB1));
                datl= (~inp(PA1));
                outp(PC1,RESET);
                dath -= num;
                datl -= num;
                d5=dath/16;
                d5=dath-d5*16;
                d6=datl/16;
                d7=datl-d6*16;
        }

/* clear the address and data output ports */

PQ:     outp(PC1,RESET);
        outp(PA2,RESET);
        outp(PB2,RESET);

}

/*-------------------------------------------------------------*/
AB_offmode()
{
        int address, dath, datl;

/* Set address for off mode -- octal 0101. */

        address = 0101;
        dath = 0x0;
        datl = 0x0;

/* Command the mode.  */

        outp(PC1,~address);
        outp(PA2,~datl);
        outp(PB2,~dath);

        while(flag == flagi)
        {
```

```
                flag = inp(PC2);
        }
        flag = flagi;

        outp(PC2,STROBE);
        outp(PC2,RESET);

/* Command mode read from mode registers -- octal 005. */

        address = 005;
        outp(PC1,~address);
        outp(PA2,~datl);
        outp(PB2,~dath);

        while(flag == flagi)
        {
                flag = inp(PC2);
        }
        flag = flagi;

        outp(PC2,STROBE);
        outp(PC2,RESET);

/* Clear address ports and data ouput ports. */

        outp(PC1,RESET);
        outp(PA2,RESET);
        outp(PB2,RESET);

}
```

KOHONEN1.C

- A "1-D NEIGHBORHOOD" VERSION

- INPUT

  (i) $(\theta, \varphi)_k$ File

  (ii) Initial Weights File

- OUTPUT

  Final Weights File

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#define IPRINT 100
main()
{
  char str[15];
  long iterations,iter;
  int iprint;
  int proc,npts,i,j,k,l,match,width;
  int row,col,row_match,col_match,side,row_start,col_start;
  float xf,yf,t1f,t2f;
  double *x,*y,*m1,*m2,*t1,*t2,*n1,*n2,dth1,dth2;
  double alpha,dold,dnew,a;

  FILE *fptr;

/***********************************************

       enter input data

***********************************************/

  printf("enter the number of pe's ");
  gets(str);
  proc=atoi(str);

  printf("enter number of iterations ");
  gets(str);
  iterations=atol(str);

  printf("enter the size of the nbhd (index width) ");
  gets(str);
  width=atoi(str);

  printf("enter the starting learning gain alpha ");
  gets(str);
  alpha=atof(str);

  /***********************************************
       read the input patterns
  ***********************************************/

  printf("enter number of points to input ");
  gets(str);
  npts=atoi(str);

  t1=(double *) calloc(npts,sizeof(double));
  t2=(double *) calloc(npts,sizeof(double));

  printf("enter the name of the input points file ");
  gets(str);
  fptr=fopen(str,"r");
  for(i=0;i<npts;i++)  {
    fscanf(fptr,"%g %g \n",&xf,&yf);
    t1[i]=xf;
    t2[i]=yf;
  }
  fclose(fptr);
```

```
  n1=(double *) calloc(proc,sizeof(double));
  n2=(double *) calloc(proc,sizeof(double));

  randomize();
/*
  for (i=0;i<npts;i++ ) {
    t1[i]=((double) rand()) / ((double) RAND_MAX);
    t2[i]=((double) rand()) / ((double) RAND_MAX);
  }
*/


  /*************************************************
        randomize the weights
  *************************************************/

  printf("Do you want to input an initial weight file? (y/n)");
  gets(str);
  if(strcmp(str,"y")==0) {
    printf("enter the name of the weight file ");
    gets(str);
    fptr=fopen(str,"r");
    for(i=0;i<proc;i++)  {
      fscanf(fptr,"%g %g \n",&xf,&yf);
      n1[i]=xf;
      n2[i]=yf;
    }
    fclose(fptr);
  }
  else {

    for(i=0;i<proc;i++)  {
      n1[i]=((double) rand()) / ((double) RAND_MAX);
      n2[i]=((double) rand()) / ((double) RAND_MAX);
    }
  }

  /*************************************************
        start learning algorithm
  *************************************************/
    a=alpha;
    iprint=0;
    for(iter=0;iter<iterations;iter++)  {
      i=random(npts);
/*********************************************************
      t1[i]=((double) rand()) / ((double) RAND_MAX);
      t2[i]=((double) rand()) / ((double) RAND_MAX);
*********************************************************/
      a=alpha - ((double) iter / (double) iterations) *alpha;

        /*****************
         similarity match
        *****************/

        dold=(t1[i]-n1[0])*(t1[i]-n1[0]) + (t2[i]-n2[0])*(t2[i]-n2[0]);
        match=0;
        for(j=1;j<proc;j++)  {
          dnew=(t1[i]-n1[j])*(t1[i]-n1[j]) + (t2[i]-n2[j])*(t2[i]-n2[j]);
```

```
          if(dnew<dold) {
            match=j;
            dold=dnew;
          }
        }

        /******************
             updating
        ******************/

        for(j=0;j<width;j++)  {
          k=match-(width/2) + j;
          if((k>=0) && (k<=proc))  {
            n1[k]+= a*( t1[i] - n1[k] );
            n2[k]+= a*( t2[i] - n2[k] );
          }
        }

/*********************************************
        side=sqrt((double) proc);
        row_match=match/side;
        col_match=match-(row_match*side);
        row_start=row_match  - (width/2);
        col_start=col_match - (width/2);
        for(j=0;j<width;j++)  {
          for(l=0;l<width;l++)  {
            row=row_start+l;
            col=col_start+j;
            if( (row>=0) && (row<side)  &&
                (col>=0) && (col<side) )  {

              k=row*side + col ;
              if((k>=0) && (k<=proc))  {
                n1[k]+= a*( t1[i] - n1[k] );
                n2[k]+= a*( t2[i] - n2[k] );
              }
            }
          }
        }
*********************************************/
        iprint++;
        if(iprint>IPRINT) {
          printf("======> just finished iteration %d \n",iter);
          iprint=0;
        }
  }
/*********************************************
        output the weights
*********************************************/

  printf("enter name of output file ");
  gets(str);
  fptr=fopen(str,"w");
  for(i=0;i<proc;i++)  {
    fprintf(fptr,"%g %g \n",n1[i],n2[i]);
  }
  fclose(fptr);
}
```

KOHONEN2.C

- A "2-D NEIGHBORHOOD" VERSION

- INPUT

  (i) $(\theta, \varphi)_k$ File

  (ii) Initial Weights File

- OUTPUT

  Final Weights File

```
 #include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#define IPRINT 100
main()
{
  char str[15];
  long iterations,iter;
  int iprint;
  int proc,npts,i,j,k,l,match,width;
  int row,col,row_match,col_match,side,row_start,col_start;
  float xf,yf,t1f,t2f;
  double *x,*y,*m1,*m2,*t1,*t2,*n1,*n2,dth1,dth2;
  double alpha,dold,dnew,a;

  FILE *fptr;

/***********************************************

       enter input data

***********************************************/

  printf("enter the number of pe's ");
  gets(str);
  proc=atoi(str);

  printf("enter number of iterations ");
  gets(str);
  iterations=atol(str);

  printf("enter the size of the nbhd (index width) ");
  gets(str);
  width=atoi(str);

  printf("enter the starting learning gain alpha ");
  gets(str);
  alpha=atof(str);

  /***********************************************
       read the input patterns
  ***********************************************/

  printf("enter number of points to input ");
  gets(str);
  npts=atoi(str);

  x=(double *) calloc(npts,sizeof(double));
  y=(double *) calloc(npts,sizeof(double));
  t1=(double *) calloc(npts,sizeof(double));
  t2=(double *) calloc(npts,sizeof(double));

  printf("enter the name of the input points file ");
  gets(str);
  fptr=fopen(str,"r");
  for(i=0;i<npts;i++)  {
    fscanf(fptr,"%g %g \n",&xf,&yf);
    t1[i]=xf;
    t2[i]=yf;
```

```
  }
  fclose(fptr);


  m1=(double *) calloc(proc,sizeof(double));
  m2=(double *) calloc(proc,sizeof(double));
  n1=(double *) calloc(proc,sizeof(double));
  n2=(double *) calloc(proc,sizeof(double));

  randomize();
/*
  for (i=0;i<npts;i++ ) {
    t1[i]=((double) rand()) / ((double) RAND_MAX);
    t2[i]=((double) rand()) / ((double) RAND_MAX);
    x[i]=cos(t1[i]);
    y[i]=sin(t1[i]);
  }
*/


  /*************************************************
        randomize the weights
  *************************************************/

  printf("Do you want to input an initial weight file? (y/n)");
  gets(str);
  if(strcmp(str,"y")==0) {
    printf("enter the name of the weight file ");
    gets(str);
    fptr=fopen(str,"r");
    for(i=0;i<proc;i++)  {
      fscanf(fptr,"%g %g %g %g \n",&xf,&yf,&t1f,&t2f);
      m1[i]=xf;
      m2[i]=yf;
      n1[i]=t1f;
      n2[i]=t2f;
    }
    fclose(fptr);
  }
  else {

    for(i=0;i<proc;i++)  {
      m1[i]=((double) rand()) / ((double) RAND_MAX);
      m2[i]=((double) rand()) / ((double) RAND_MAX);
      n1[i]=((double) rand()) / ((double) RAND_MAX);
      n2[i]=((double) rand()) / ((double) RAND_MAX);
    }
  }

  /*************************************************
        start learning algorithm
  *************************************************/
    a=alpha;
    iprint=0;
    for(iter=0;iter<iterations;iter++)  {
      i=random(npts);
/*******************************************************
      t1[i]=((double) rand()) / ((double) RAND_MAX);
      t2[i]=((double) rand()) / ((double) RAND_MAX);
      x[i]=cos(t1[i]);
```

```
      y[i]=sin(t1[i]);
*****************************************************/
      a=alpha - ((double) iter / (double) iterations) *alpha;

        /*****************
         similarity match
        ******************/

/*      dold=(x[i]-m1[0])*(x[i]-m1[0]) + (y[i]-m2[0])*(y[i]-m2[0]);  */
        dold=(t1[i]-n1[0])*(t1[i]-n1[0]) + (t2[i]-n2[0])*(t2[i]-n2[0]);
        match=0;
        for(j=1;j<proc;j++)  {
/*        dnew=(x[i]-m1[j])*(x[i]-m1[j]) + (y[i]-m2[j])*(y[i]-m2[j]); */
          dnew=(t1[i]-n1[j])*(t1[i]-n1[j]) + (t2[i]-n2[j])*(t2[i]-n2[j]);
          if(dnew<dold) {
            match=j;
            dold=dnew;
          }
        }

        /******************
             updating
        ******************/
/*********************************************************************

        for(j=0;j<width;j++)  {
          k=match-(width/2) + j;
          if((k>=0) && (k<=proc))  {
            m1[k]+= a*( x[i] - m1[k] );
            m2[k]+= a*( y[i] - m2[k] );
            n1[k]+= a*( t1[i] - n1[k] );
            n2[k]+= a*( t2[i] - n2[k] );
          }
        }

*********************************************************************/
        side=sqrt((double) proc);
        row_match=match/side;
        col_match=match-(row_match*side);
        row_start=row_match  - (width/2);
        col_start=col_match - (width/2);
        for(j=0;j<width;j++)  {
          for(l=0;l<width;l++)  {
            row=row_start+l;
            col=col_start+j;
            if( (row>=0) && (row<side)  &&
                (col>=0) && (col<side) )  {

              k=row*side + col ;
              if((k>=0) && (k<=proc))  {
                m1[k]+= a*( x[i] - m1[k] );
                m2[k]+= a*( y[i] - m2[k] );
                n1[k]+= a*( t1[i] - n1[k] );
                n2[k]+= a*( t2[i] - n2[k] );
              }
            }
          }
        }
        iprint++;
        if(iprint>IPRINT) {
```

```
          printf("======> just finished iteration %d \n",iter);
          iprint=0;
        }
  }
/*********************************************
        output the weights
*********************************************/

  printf("enter name of output file ");
  gets(str);
  fptr=fopen(str,"w");
  for(i=0;i<proc;i++)  {
    fprintf(fptr,"%g %g %g %g \n",m1[i],m2[i],n1[i],n2[i]);
  }
  fclose(fptr);
}
```

RAD.C

- Radial Basis Function Interpolation Function

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TRUE 1
#define FALSE 0
#define SMALL .00001
#define TOL 0.0
#define SCALE ( (double) (2*1024*1024*1024-1) ) /* i.e. 2^31-1 */
#include "param.h"
unsigned seed = 1; /* random number seed */
double inner_prod();
double a[N][N]; /* matrix storage */
double x[N], y[N], z[N]; /* vector storage */
double c[N]; /* coefficient storage */
double dist;
double store;
float xf,yf,zf;

FILE *fopen(), *actual, *approx1, *approx2, *err1, *err2, *wvinput;

main(argc,argv)
{
        register int i, j;
        double get_rand(), f(), g(), error(), app();
        double r, xval, yval, error1max, error2max,xmin,xmax,ymin,ymax;
        void bndlu(), bndslv();
        time_t t;

        srand((unsigned) time(&t));

        if ((actual=fopen("actual.dat","w")) == NULL)
                fprintf (stderr,"can't open actualdata\n");
        if ((approx1=fopen("approx1.dat","w")) == NULL)
                fprintf (stderr,"can't open approx1data\n");
        if ((approx2=fopen("approx2.dat","w")) == NULL)
                fprintf (stderr,"can't open approx2data\n");
        if ((err1=fopen("error1.dat","w")) == NULL)
                fprintf (stderr,"can't open error1\n");
        if ((err2=fopen("error2.dat","w")) == NULL)
                fprintf (stderr,"can't open error2\n");
        if ((wvinput=fopen("wavinp.dat","r"))==NULL)
                fprintf (stderr,"can't open wavinp.dat\n");

        for(i=0;i<N;i++){
/*************************************************************
                x[i]=XMIN + get_rand()*(XMAX-XMIN);
                y[i]=YMIN + get_rand()*(YMAX-YMIN);
                z[i]=g(x[i],y[i]);
*************************************************************/
          fscanf(wvinput,"%g %g %g \n",&xf,&yf,&zf);
          x[i]=xf;
          y[i]=yf;
          z[i]=zf;

          if(i==0){
            xmin=xf;
            ymin=yf;
            xmax=xf;
```

```
            ymax=yf;
          }
          else {
            if(xf > xmax)xmax=xf;
            if(xf < xmin)xmin=xf;
            if(yf > ymax)ymax=yf;
            if(yf < ymin)ymin=yf;
          }
        }
        find_sep();

        dist=1.0;
        for(i=0;i<N;i++)
                for(j=0;j<N;j++){
                        r=sqrt( (x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j]))
                        a[i][j]=f(dist,r,ALPHA);
                }
        if(N<50){
                bndlu(a,a,a,N);
                bndslv(a,a,z,c,N);
        }
        else
                conjugate_grad(a,c,z);
        /*
        for(i=0;i<N;i++)
                printf("c[%d]=%lf\n",i,c[i]);
        */
        for(i=0;i<=M;i++){
                xval=xmin+i*(xmax-xmin)/(double)M;
                for(j=0;j<=M;j++){
                        yval=ymin+j*(ymax-ymin)/(double)M;
                        fprintf(approx1,"%lf \n",app(xval,yval));
                }
        }

        dist=1.0;
        for(i=0;i<N;i++)
                for(j=0;j<N;j++){
                        r=sqrt( (x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j]))
                        a[i][j]=f(dist,r,ALPHA);
                }
        if(N<50){
                bndlu(a,a,a,N);
                bndslv(a,a,z,c,N);
        }
        else
                conjugate_grad(a,c,z);
        for(i=0, error2max=0.0;i<=M;i++){
                xval=xmin+i*(xmax-xmin)/(double)M;
                for(j=0;j<=M;j++){
                        yval=ymin+j*(ymax-ymin)/(double)M;
                        fprintf(approx2,"%lf \n",app(xval,yval));
                }
        }
}

find_sep()
{
        register int i,j;
        double sep=XMAX-XMIN, val=XMAX-XMIN;
```

```
        for(i=0;i<N;i++)
                for(j=0;j<N;j++){
                        sep=sqrt((x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j])
                        if ( (i != j) && (sep<val)){
                                val=sep;
                        }
                }
        dist=val;
        printf("seperation= %g\n",dist);
}

double app(xval,yval)
double xval, yval;
{
                double r, val, f();
                register int j;

                for(j=0, val=0.0;j<N;j++){
                        r=sqrt( (xval-x[j])*(xval-x[j])+(yval-y[j])*(yval-y[j]))
                        val+= c[j]*f(dist,r,ALPHA);
                }
                return(val);
}

double error(xval,yval)
double xval, yval;
{
                double r, val, f();
                register int j;

                for(j=0, val=0.0;j<N;j++){
                        r=sqrt( (xval-x[j])*(xval-x[j])+(yval-y[j])*(yval-y[j]))
                        val+= c[j]*f(dist,r,ALPHA);
                }
                return(val-g(xval,yval)); /* signed error */
}

double get_rand()
{
        srand(seed);

        return( rand()/SCALE ) ;
}

double f(d,z)
double d,z;
{
        return( INTERP(d,z) );
}

double g(xval,yval)
double xval,yval;
{
        return(FUNCTION(xval,yval));
}

void bndlu(array,lower,upper,bw)
double array[N][N], lower[N][N], upper[N][N];
int bw;
```

```
{
        int i,j,k,n=N;
        int intceil(), intlower();
        double sum;

/* lower[0][0] = 1.0,  normalization assumed */
        upper[0][0] = array[0][0]; /* when array=lower=upper, overwrite */
        for(i=1;i<=intceil(bw,n-1);i++){
                upper[0][i]=array[0][i];
                lower[i][0]=array[i][0]/array[0][0];
        }
        for(i=1;i<=intceil(bw,n-1);i++){
                sum = 0.0;
                for(k = intfloor(i-bw,0);k<i;k++){
                        sum += lower[i][k]*upper[k][i];
                }
/* lower[i][i] = 1.0, normalization assumed */
                upper[i][i] = array[i][i] - sum; /* when array=lower=upper, over
                for(j=i+1;j<intceil(i+bw,n);j++){
                        sum = 0.0;
                        for(k=intfloor(i-bw,0);k<i;k++){
                                sum += lower[i][k]*upper[k][j];
                        }
                        upper[i][j] = array[i][j] - sum;
                        sum = 0.0;
                        for(k=intfloor(i-bw,0);k<i;k++){
                                sum += lower[j][k]*upper[k][i];
                        }
                        lower[j][i] = ( array[j][i] - sum )/upper[i][i];
                }
        }
}

int intfloor(x,bottom)  /* takes max of two integers */
int x,bottom;
{
        return( (x > bottom) ? x : bottom );
}

int intceil(x,top)  /* takes min of two integers */
int x,top;
{
        return( (x < top) ? x : top );
}

void bndslv(lower,upper,rhs,solution,bw)
double rhs[N], solution[N], lower[N][N], upper[N][N];
int bw;
{
        double sum;
        int i,j,k,n=N;
        int intceil(), intlower();

        for(i=0;i<n;i++){
                solution[i]=rhs[i];
        }
        for(i=1;i<n;i++){
                sum=0.0;
                for(k=intfloor(i-bw,0);k<i;k++){
                        sum = sum + lower[i][k]*solution[k];
```

```
                }
                solution[i] = solution[i] - sum;
        }
        solution[n-1] /= upper[n-1][n-1];
        for(i=n-2;i>=0;i--){
                sum = 0.0;
                for(k=i+1;k<=intceil(i+bw,n-1);k++){
                        sum = sum + upper[i][k]*solution[k];
                }
                solution[i] = (solution[i]-sum)/upper[i][i];
        }
}

/* apply conjugate-gradient (2-term) method */
/* use random number generators */

double inner_prod(v1,v2)
double v1[N], v2[N];
{
        int i;
        double inprod;

        inprod=0.0;
        for(i=0;i<N;i++)
                inprod += v1[i] * v2[i];
        return(inprod);
}

conjugate_grad(mat,vec,f)
double mat[N][N], vec[N], f[N];
{
        int i,j,k;
        double r[N],p[N],ap[N];
        double alpha,lambda,denom;

        for(i=0;i<N;i++){
                r[i]=0.0;
                for(j=0;j<N;j++)
                        r[i] += -mat[i][j] * vec[j];
                r[i] += f[i];
        }
        for(i=0;i<N;i++)
                p[i]=r[i];
        for(i=0;i<N;i++){
                ap[i]=0.0;
                for(j=0;j<N;j++)
                        ap[i] += mat[i][j]*p[j];
        }
        denom=inner_prod(p,ap);
        if(fabs(denom) < TOL){
                printf("denominator too small!\n");
                printf("x = %lf \n");
                exit(-1);
        }
        else
                lambda= inner_prod(r,r)/ denom;

        for(i=0;i<N;i++)
                vec[i] += lambda*p[i];
```

```
        /* begin the main iteration */
        for(k=0;k< itmax ;k++){
                denom = inner_prod(p,ap);
                for(i=0;i<N;i++){
                        r[i]=0.0;
                        for(j=0;j<N;j++)
                                r[i] += -mat[i][j] * vec[j];
                        r[i] += f[i];
                }

                if(fabs(denom) < TOL){
                        printf("denominator too small!\n");
                        printf("x = %lf \n");
                        exit(-1);
                }
                else
                        alpha = -inner_prod(r,ap)/ denom ;

                for(i=0;i<N;i++)
                        p[i]=r[i]+alpha*p[i];
                for(i=0;i<N;i++){
                        ap[i]=0.0;
                        for(j=0;j<N;j++)
                                ap[i] += mat[i][j]*p[j];
                }
                lambda=inner_prod(p,r)/inner_prod(p,ap);
                for(i=0;i<N;i++)
                        vec[i] += lambda*p[i];
               printf("L2 norm of residual=%lf\n",sqrt(inner_prod(r,r)));
        }
}
```

COLEVAL.C

- Colinearity Equations Evaluation Functions

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
  float vx,vy,x,y,z,x0,y0,xc,yc,zc,phi,theta,psi,f;
  float phis,phie,ths,the;
  char str[80];
  FILE *fptr0,*fptr1,*fptr2;
  int i,j,side;

  printf("enter the grid size ");
  gets(str);
  side=atoi(str);

  printf("enter x0 ");
  gets(str);
  x0=atof(str);

  printf("enter y0 ");
  gets(str);
  y0=atof(str);

  printf("enter the focal length f");
  gets(str);
  f=atof(str);

  printf("enter the object point X ");
  gets(str);
  x=atof(str);

  printf("enter the object point Y ");
  gets(str);
  y=atof(str);

  printf("enter the object point Z ");
  gets(str);
  z=atof(str);

  xc=0.;
  yc=0.;
  zc=0.;
  psi=0.;

  printf("enter the name of the phi-theta input file ");
  gets(str);
  fptr0=fopen(str,"r");

  printf("enter the name of the x-y output file ");
  gets(str);
  fptr1=fopen(str,"w");

  printf("enter the name of the x-y-phi-theta output file ");
  gets(str);
  fptr2=fopen(str,"w");

  for(i=0;i<side;i++)  {
    for(j=0;j<side;j++)  {
      fscanf(fptr0,"%g %g ",&phi,&theta);
      colinearity(&vx,&vy,x,y,z,x0,y0,xc,yc,zc,phi,theta,psi,f);
```

```
      fprintf(fptr1,"%g %g \n",vx,vy);
      fprintf(fptr2,"%g %g %g %g \n",vx,vy,phi,theta);
    }
  }
  fclose(fptr0);
  fclose(fptr1);
  fclose(fptr2);
}

#include <math.h>
int colinearity(vx,vy,x,y,z,x0,y0,xc,yc,zc,phi,theta,psi,f)
float *vx,*vy,x,y,z,x0,y0,xc,yc,zc,phi,theta,psi,f;
{
  float cpsi[3][3],cth[3][3],cphi[3][3];
  float c[3][3],num,den;
  int i,j,k,l;

  cpsi[0][0]=1.;
  cpsi[0][1]=0.;
  cpsi[0][2]=0.;
  cpsi[1][0]=0.;
  cpsi[1][1]=cos((double) psi);
  cpsi[1][2]=sin((double) psi);
  cpsi[2][0]=0.;
  cpsi[2][1]=-sin((double) psi);
  cpsi[2][2]=cos((double) psi);

  cth[0][0]=cos((double) theta);
  cth[0][1]=0.;
  cth[0][2]=-sin((double) theta);
  cth[1][0]=0.;
  cth[1][1]=1.;
  cth[1][2]=0.;
  cth[2][0]=sin((double) theta);
  cth[2][1]=0.;
  cth[2][2]=cos((double) theta);

  cphi[0][0]=cos((double) phi);
  cphi[0][1]=sin((double) phi);
  cphi[0][2]=0.;
  cphi[1][0]=0.;
  cphi[1][1]=-sin((double) phi);
  cphi[1][2]=cos((double) phi);
  cphi[2][0]=0.;
  cphi[2][1]=0.;
  cphi[2][2]=1.;

  for(i=0;i<3;i++)  {
    for(j=0;j<3;j++)  {
      c[i][j]=0;
      for(k=0;k<3;k++)  {
        for(l=0;l<3;l++)  {
          c[i][j]+= cpsi[i][k]*cth[k][l]*cphi[l][j];
        }
      }
    }
  }

  num=c[0][0]*(x-xc) + c[0][1]*(y-yc) + c[0][2]*(z-zc);
  den=c[2][0]*(x-xc) + c[2][1]*(y-yc) + c[2][2]*(z-zc);
```

```
  *vx=x0 - f*num/den;

  num=c[1][0]*(x-xc) + c[1][1]*(y-yc) + c[1][2]*(z-zc);
  den=c[2][0]*(x-xc) + c[2][1]*(y-yc) + c[2][2]*(z-zc);
  *vy=y0 - f*num/den;
}
```

COGEN.C

- Colinearity Equation I/O Generator

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
  float vx,vy,x,y,z,x0,y0,xc,yc,zc,phi,theta,psi,f;
  float phis,phie,ths,the;
  char str[80];
  FILE *fptr,*fptr0;
  int i,j,side;

  printf("enter the starting phi value ");
  gets(str);
  phis=atof(str);

  printf("enter the ending phi value ");
  gets(str);
  phie=atof(str);

  printf("enter the starting theta value ");
  gets(str);
  ths=atof(str);

  printf("enter the ending theta value ");
  gets(str);
  the=atof(str);

  printf("enter the grid size ");
  gets(str);
  side=atoi(str);

  printf("enter x0 ");
  gets(str);
  x0=atof(str);

  printf("enter y0 ");
  gets(str);
  y0=atof(str);

  printf("enter the focal length f");
  gets(str);
  f=atof(str);

  printf("enter the object point X ");
  gets(str);
  x=atof(str);

  printf("enter the object point Y ");
  gets(str);
  y=atof(str);

  printf("enter the object point Z ");
  gets(str);
  z=atof(str);

  xc=0.;
  yc=0.;
  zc=0.;
  psi=0.;
```

```
  printf("enter the name of the x-y output file ");
  gets(str);
  fptr=fopen(str,"w");

  printf("enter the name of the x-y-phi-theta output file ");
  gets(str);
  fptr0=fopen(str,"w");

  for(i=0;i<side;i++)  {
    theta=ths + ((float)i)*(the-ths)/( (float) side );
    for(j=0;j<side;j++)  {
      phi=phis + ((float)j)*(phie-phis)/( (float) side );
      colinearity(&vx,&vy,x,y,z,x0,y0,xc,yc,zc,phi,theta,psi,f);
      fprintf(fptr,"%g %g \n",vx,vy);
      fprintf(fptr0,"%g %g %g %g \n",vx,vy,phi,theta);
    }
  }
}

#include <math.h>
int colinearity(vx,vy,x,y,z,x0,y0,xc,yc,zc,phi,theta,psi,f)
float *vx,*vy,x,y,z,x0,y0,xc,yc,zc,phi,theta,psi,f;
{
  float cpsi[3][3],cth[3][3],cphi[3][3];
  float c[3][3],num,den;
  int i,j,k,l;

  cpsi[0][0]=1.;
  cpsi[0][1]=0.;
  cpsi[0][2]=0.;
  cpsi[1][0]=0.;
  cpsi[1][1]=cos((double) psi);
  cpsi[1][2]=sin((double) psi);
  cpsi[2][0]=0.;
  cpsi[2][1]=-sin((double) psi);
  cpsi[2][2]=cos((double) psi);

  cth[0][0]=cos((double) theta);
  cth[0][1]=0.;
  cth[0][2]=-sin((double) theta);
  cth[1][0]=0.;
  cth[1][1]=1.;
  cth[1][2]=0.;
  cth[2][0]=sin((double) theta);
  cth[2][1]=0.;
  cth[2][2]=cos((double) theta);

  cphi[0][0]=cos((double) phi);
  cphi[0][1]=sin((double) phi);
  cphi[0][2]=0.;
  cphi[1][0]=0.;
  cphi[1][1]=-sin((double) phi);
  cphi[1][2]=cos((double) phi);
  cphi[2][0]=0.;
  cphi[2][1]=0.;
  cphi[2][2]=1.;

  for(i=0;i<3;i++)  {
    for(j=0;j<3;j++)  {
      c[i][j]=0;
```

```
      for(k=0;k<3;k++)  {
        for(l=0;l<3;l++)  {
          c[i][j]+= cpsi[i][k]*cth[k][l]*cphi[l][j];
        }
      }
    }
  }

  num=c[0][0]*(x-xc) + c[0][1]*(y-yc) + c[0][2]*(z-zc);
  den=c[2][0]*(x-xc) + c[2][1]*(y-yc) + c[2][2]*(z-zc);
  *vx=x0 - f*num/den;

  num=c[1][0]*(x-xc) + c[1][1]*(y-yc) + c[1][2]*(z-zc);
  den=c[2][0]*(x-xc) + c[2][1]*(y-yc) + c[2][2]*(z-zc);
  *vy=y0 - f*num/den;
}
```