8-1-1981

Real Time Control of a Robot Tacticle Sensor

Jeffrey A. Wolfeld
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-81-04.

Real Time Control of a Robot Tacticle Sensor

Abstract

The goal of the Experimental Sensory Processor project is to build a system which employs both visual and tactile senses, and then explore their interaction in a robotic environment. Here we describe the software involved in the low level control of the tactile branch of this system, and present results of some simple experiments performed with a prototype tactile sensor.

Disciplines

Robotics

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-81-04.

UNIVERSITY OF PENNSYLVANIA
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

REAL TIME CONTROL OF A ROBOT TACTILE SENSOR

Jeffrey A. Wolfeld

Philadelphia, Pennsylvania

August, 1981

A thesis presented to the Faculty of Engineering and Applied Science in partial fulfillment of the requirements for the degree of Master of Science in Engineering for graduate work in Computer and Information Science.

Ruzena Bajcsy

Aravind K. Joshi

The work reported here was supported in part by NSF grant number MCS-78-07466.

REAL TIME CONTROL OF A ROBOT TACTILE SENSOR

Jeffrey A. Wolfeld

Philadelphia, Pennsylvania

August 1981

Abstract

The goal of the Experimental Sensory Processor project is to build a system which employs both visual and tactile senses, and then explore their interaction in a robotic environment. Here we describe the software involved in the low level control of the tactile branch of this system, and present results of some simple experiments performed with a prototype tactile sensor.

Acknowledgments

I would like to thank the following people:

Jim Korein, my office mate, with whom I had many fascinating discussions between 9:00 and 5:00 on weekdays;

Gerry Radack, who occasionally dragged me away from my terminal in order to play music;

Clayton Dane, who helped keep my feet on the ground;

Jeff Shrager, without whom I might never have gotten past the Abstract;

Taylor Adair, who kept the computer running when it really wanted to crash;

Ira Winston, who served as the local oracle;

Jack Rebman of the Lord Corporation, without whom this thesis would have been entirely speculation;

David Brown, who got me into this mess in the first place;

and my advisor, Ruzena Bajcsy, mother to us all.

Table of Contents PAGE

Chapter 1: Introduction

1.1 Motivation

Artificial Intelligence researchers have worked extensively with vision systems in an attempt to give computers, and eventually robots, a sense of sight. A great deal of this research has been directed toward overcoming certain basic inadequacies in our current technology. For example, imperfect light sensors dictate that noise must be eliminated or tolerated. Insufficient spatial resolution requires routines which will interpolate below the pixel level.

One of the most important problems is that a camera produces a two-dimensional image of a three-dimensional scene. This invalidates an assumption which one would like to rely upon -- that two adjacent points in the image are adjacent in the scene. Therefore, substantial effort has been devoted to reproducing 3-D data from one or several visual images. Tactile sensors can be used to aid the process.

An imaging tactile sensor, by its very nature, does not have the problem. Since it produces a two-dimensional image

of a two-dimensional scene, it does not provide as much information, but it yields useful information clearly, without the need for complicated heuristics.

We can take this one step further. Suppose a tactile sensor is mounted on some kind of computer controlled 3-D positioning device. Then, by moving the sensor to different points on a target object, the computer can actually obtain 3-D data directly, and much more selectively. If this information is used to supplement and augment visual data, a great deal of processing may be avoided.

One can come up with many other uses for varying kinds of tactile sensors. Briot [BRIOT-79] demonstrated that tactile sensors mounted on the fingers of a robot hand can be used to determine the position, orientation, and perhaps even the identity of an object which it has grasped. He also showed that a grid of pressure sensitive sites on a table can tell a robot the location, orientation, and again, the identity of a part. It should be possible with multi-valued pressure sensors, as opposed to binary sensors, to determine the mass of the object. When the angle is small, a tactile sensor can be used to compute the angle between it and the object being grasped, possibly with a view toward improving the grip. Also, if the device is sensitive enough, it can be an invaluable aid to a robot attempting to grasp a fragile object without breaking it. Finally, a tactile sensor makes it possible to incorporate the properties of surface texture

and resilience into the object recognition process.

1.2 Project Overview

The design and development of the tactile system has proceeded with two different sensors in mind. Unfortunately, there are so many disparities between the two that we had difficulty keeping the system general enough to handle both. Let this serve as a demonstration of the variety of characteristics that must be considered for a given application.

The first sensor is about five inches long, with an octagonal cross section about 3/4 inches in diameter. Each of the eight rectangular faces is connected to a tapered piece, which is in turn connected to a common tip piece. There are a total of 133 sensitive sites -- 16 on each main face, one on each alternate taper, and one on the tip. Because of the the vague resemblance, we will refer to this sensor as the Finger.

The second sensor, the Pad, is a flat rubber square about two and one half inches on a side. An 8 x 8 grid of conical protrusions identify the 64 pressure sensitive sites. The pad is mounted on a square metal piece, about three and one half inches on a side, which is in turn connected to another similar piece by four metal posts. These posts have strain gauges on them which measure the force parallel to the object's surface.

Initially, we only considered the finger. Because of its shape and organization, the sensor is best suited to applications involving probing and tracing. This includes testing for resiliency, examining surface texture, and tracing cross-sections of an object. In our view, texture would be thought of as a kind of microscopic contour, while the cross-section tracings would yield a macroscopic contour. Taken together, we would be able to acquire an extremely detailed description of very selective parts of the object in question.

Unfortunately, this rather vague idea has not been developed. We have instead dealt with the two descriptions independently with the assumption that they can both be incorporated into a general object recognition system.

For his Master's Thesis, David Brown [BROWN-80] developed a three-dimensional positioning device for the finger. Basically, it is a square horizontal metal frame mounted on four legs. Moving forward and backward on this is a second, vertical square frame. A vertical track rides left to right on that, and a rod moves up and down in the track. The finger would be mounted with its tip downward at the bottom of the rod.

Thus, we have three degrees of freedom -- the X, Y and Z axes -- each positioned by a stepper motor driving a lead screw. This gives us the capability of examining, from the top, any object or objects placed on a table below the

horizontal frame, in a total working volume of about 18 cubic inches. Since the degrees of freedom are strictly positional, as opposed to rotational, we are not capable of reaching under an overhanging lip, or sideways below a covering section. This places certain restrictions on the kind of object we can examine. If we think of the horizontal axes as X and Y, then the object must be describable as a strict function of those two variables. Needless to say, this is not a robot arm, but we felt it would suffice, temporarily at least, for our research.

The positioning device and tactile sensor are directly controlled by a pair of Z80 microprocessors, which are in turn under the command of a PDP-11/60 minicomputer. Of the Z80's, one (the Motor Control Processor, MCP) is responsible for driving and positioning the stepper motors, and the other (the Tactile Sensing Processor, TSP) is dedicated to tactile data acquisition and compression. The MCP and TSP communicate with each other via a 14-bit wide parallel data path. The PDP-11/60 issues high level commands, and receives positional information, through a serial connection to the MCP. Finally, tactile data is passed to the 11/60 through a DMA link from the TSP.

One of the aforementioned high level commands would request the microprocessors to trace the cross-section of an object in any arbitrary plane in space, passing the sequence of 3-D coordinates back to the host computer. A great deal

of thought went into the implementation of this command, and it is, to some extent, responsible for the architecture described above. The procedure will be described in detail in a later section. It is a good example of how tactile sensory feedback can be used in a real time, closed loop fashion.

The finger was designed and fabricated at L.A.A.S., the major robotics establishment of the French government. Because of a severe lack of communication, many of the finger's details were not known to us when the software was being designed. This had a positive affect in that we were forced to be as general as possible. However, due to a number of unexpected delays, we still do not have the finger in our possession.

We arranged to borrow the pad sensor from Lord Corporation in Erie, Penna.* They traditionally deal with blending rubbers and bonding rubber to metal. This sensor, still in the prototype stage, is an attempt to expand their business.

At any rate, we had the pad sensor in our possession for three very long days. In preparation for that ordeal, we planned a number of different experiments. The Lord people were very helpful in this, and they provided us with the appropriate wooden test objects.

---

* Lord has since moved to Cary, South Carolina.

The characteristics of the pad sensor are very different than those of the finger. In particular, there is only one sensitive face. This makes the pad much less suited to contour tracing. We therefore decided to concentrate on some of the other aspects of tactile sensing -- dynamic texture analysis, static pattern recognition, and measurement of small angles between the object and sensor surfaces.

The ensuing sections will describe in detail the work performed.

Chapter 2: The Proposed Microprocessor Software

In anticipation of the arrival of the finger, a great deal of software was planned. Then, when the delays became apparent, work on those aspects not directly applicable to the pad sensor screeched to a halt. As a result, some of the design described here has not yet been implemented. In a later section we will discuss in detail exactly what the existing software does.

One of the important features of the Experimental Sensor Processor is its delegation of low level tasks to other processors. This helps to diminish the computational load on the host pdp-11/60. The tactile branch, in keeping with this principle, would have a set of commands which could be invoked by the host to perform various I/O and timing intensive operations, or functions involving real time feedback. Following are some of the commands that were considered:

1. Reset the machine.
2. Move to absolute coordinates (x, y, z), stop on collision with an object. This can be used as a "find something in this direction" command.

3. Scan Cross-section -- Trace the contour of an object in an arbitrary plane in 3-space. Returns to the host a list of step vectors describing the finger's path.

4. Local Texture -- Trace around a small circle on the surface of an object and produce a description of the texture. This could be in terms of degree of roughness, degree of compliance, or something as crude as a list of pressure values for each point in the path.

5. Search (in an as yet unspecified manner) for either a concave or a convex edge. It is assumed that the finger is already in contact with a surface.

6. Follow the contour of a concave or convex edge. Passes a list of step vectors to the host describing the finger's path.

The first command, Reset, is trivial. It simply involves the reinitialization of variables. The move command, due to its fundamental nature, has been implemented for use with the pad sensor. The cross-sectional scan command has received a great deal of attention, but has not been completely implemented because of its incompatibility with a single-face sensor. The final three commands, Local Texture, Find Edge, and Follow Edge, have to date received very little serious consideration. They are quite tentative, and may never be implemented.

2.1 Processors

As described in other sections of this thesis, the tactile branch consists of two microprocessors, the Tactile Sensing Processor (TSP), and the Motor Control Processor (MCP). A different program runs in the firmware of each

processor. Both are entirely interrupt driven using the Z-80 vectored interrupt system. From the host computer's point of view, the TSP provides data for texture analysis, and the MCP provides data for contour analysis.

2.1.1 Tactile Sensing Processor

The TSP program consists of a single loop in which each of the sensors is interrogated for its 8-bit pressure value. Each value is thrown into one of three categories with respect to a low and a high threshold. the category indicates whether the sensor is not touching anything, is in contact with an object, or is pressing the object too hard.* The sensors are then grouped by finger face, and a face status is computed for each face using the following rules:

> If any sensor is over range, the face is over range;
> If all sensors are below range, the face is below range;
> Otherwise, the face is within range.

If there were any face status changes since the last pass, the Motor Control Processor is informed.

It is worth noticing that this condensation algorithm is independent of the particular organization of the finger. The number of faces, the faces' orientations, and even the

---

* We hope that the sensors have enough compliance of their own so we can arrange the thresholds successfully. We would like to guarantee that for any movement toward an object, there is at least one position in which the leading sensor is "in contact" before it exceeds the upper threshold.

mapping of sensor number to face number are stored in tabular form, and may be altered according to the parameters of a different sensor. It will be obvious later that the more faces we have, the easier it is to keep in contact with an object. In the ideal case, we would like a hemispherical finger with many sensors, each on its own face. Such an organization can be accommodated just as well as the current finger.

In addition to providing this condensed status information for the sister processor, the TSP must send some data to the host, for the texture analysis. How much data does the host need? If we send it all we can -- 133 8-bit bytes per step, 125 steps per second -- we would need the equivalent of 20 9600 baud serial communication lines to handle the load! The bottleneck is removed by using a Direct Memory Access (DMA) interface. But even so, we cannot expect the PDP-11/60 to analyze data arriving at such an incredible rate, and still be able to keep up with the other sensory branches, and perform the higher level recognition tasks at the same time. It simply does not have the computational power.

The answer, of course, is to filter or condense the data before sending it. We have several possibilities in mind. First, a sensor is only considered valid if its pressure value is "within range". This filter is always in effect. Other possibilities include averaging sensor

readings over time and only reporting after a fixed number of steps, or combining somehow the readings from all sensors on each face which is "within range" to produce a single face pressure value. A final possibility is to arrive at some kind of measure of roughness for the surface under consideration, and only pass that number back to the host computer. This decision has not been made.

2.1.2 Motor Control Processor

The Motor Control Processor's basic job is to control and coordinate the three stepping motors which position the finger. When it is necessary that the host computer know the path that the finger follows during the execution of a command, the MCP provides it.

Steps are taken in a synchronous fashion. That is, if the step rate is set to 125 steps per second (the default case), the processor is interrupted every eight milliseconds to determine which motors are to be stepped, and in which direction.

So, after each interval, the MCP may pulse any combination of the three motors, and each can be in one of two directions. This leads to 26 possible directions in which a single step can move (ignoring the case where no step is taken at all). We represent this direction as a

6-bit "step vector", organized as follows:

bit 5	4	3	2	1	0
Z direction	Z step	Y direction	Y step	X direction	X step

Since this fits easily in an 8-bit byte, it is very convenient now for the MCP to give a path to the host computer. It simply sends a one-byte step vector over the serial line for each step taken. The host collects the sequence of step vectors in a buffer, and the exact path can be reconstructed very quickly at any time.

There are, of course, situations in which it is necessary to give an absolute coordinate. For example, when the absolute move command is aborted due to collision with an object, it is necessary to inform the host what the new position is. A mechanism is provided for this, too.

Notice that the MCP returns (effectively) a sequence of points. It does not try to fit them to curves, surface patches, generalized cylinders, etc. This is left to the host computer. It is unreasonable to expect an 8-bit microprocessor which lacks even a multiply instruction to do these in real time.

When moving from one position to another in 3-space, it is desirable to do so in a straight line. This requires varying the speeds of the individual motors so that they all arrive at their destinations simultaneously. The following

example shows how we would like to arrange the steps in a sample situation.

	A steps	B desired time between steps
X	17	9.88 milliseconds
Y	21	8.00 milliseconds
Z	5	33.6 milliseconds

The values in column B were arrived at by dividing the column A values into the greatest column A value, and multiplying the result by 8 millisecs. (8 millisecs is the speed at which we would like the fastest motor to operate).

This is a lot of work for an 8-bit microprocessor to perform. Also, if the precision of these calculations is not great enough, it becomes virtually impossible to predict exactly where the finger will be at any given point in time.

Fortunately, the synchronous stepping scheme makes matters much simpler. The overall line of motion is a line in 3-space. This is described and stored in terms of three direction components. There are also two accumulating counters, one for the mid direction, and one for the min direction. (The mid direction is the dimension which has the second-largest number of steps to take. Min direction is defined similarly.) Both are preset to zero.

After each 8-millisecond interval, a step vector is created, and the motors are stepped accordingly. The max direction is always stepped. For each of the other two

directions, the accumulating counter is incremented by the corresponding direction component value, and the result is taken modulo the max direction component. If an overflow occured, a step is taken.

Applying the algorithm to the above example results in the following sequence of steps.

Step	X	Y	Z	Step	X	Y	Z
1		*		11		*	
2	*	*		12	*	*	
3	*	*		13	*	*	*
4	*	*		14	*	*	
5	*	*	*	15	*	*	
6		*		16		*	
7	*	*		17	*	*	*
8	*	*		18	*	*	
9	*	*	*	19	*	*	
10	*	*		20	*	*	
				21	*	*	*

When a step is taken, two corollary actions occur. First, if the MCP is providing path information, the step vector is sent to the host. Second, a termination test is made. For the absolute move command, termination occurs when the finger reaches its destination.

This command also terminates if the Tactile Sensing Processor indicates that the finger has come in contact with an object. Primarily, this is to protect the finger from damage. However, it also makes it possible for the host to say, "look in this direction for an object." In that sense, this command can be used as an object finder.

2.2 Cross-Sectional Scan Command

This command is invoked by the host to trace the contour of an object's cross-section in any arbitrary plane in 3-space.* The arguments include the coefficients a, b and c in the equation of the plane $ax + by + cz = 0$, and a pair of special 3-D points which define the search volume. The finger must already be touching an object, and the plane is assumed to pass through the finger's current position.

Consider a conical object and a slicing plane parallel to the x-y plane. The MCP will drive the finger in the plane such that it remains in contact with the surface of the cone. All the while, it passes its path back to the host. Later, the host will analyze the path, and discover that it describes a circle.

The search volume is included to limit the finger's range of motion. Suppose, for example, the host wanted to construct a 3-D bicubic surface patch. It could do this by requesting four cross-sectional scans using vertical planes whose y-z projection is a rectangle. Then it could fit curves to each of the four point sequences, and perhaps fit a patch to these four curves.

---

* My terms will be very confusing unless I define them at the outset. "Plane" generally refers to the arbitrary cross-sectional plane given by the host. "Surface" is the (possibly curved) surface of the object. "Face" refers to one of the faces of the finger on which sensors are mounted. "Search volume" means the physical volume in which the finger is allowed to move.

Unless we provide some mechanism for limiting the search space, there is no way to prevent the finger from doing a complete scan of the object's cross-section, when only a small portion of that scan is needed.

The search volume is a rectangular parallelepiped with diagonally opposed corners defined by two arbitrary points in 3-space. The arbitrary points are chosen by the host computer and passed to the MCP as arguments to this command. Very often, the points may contain special coordinate values of 0 or 'max'. These may be used to effectively leave one or more dimensions completely unconstrained.

In the surface patch example, we would like to constrain the x and y position to the projection of the four slicing planes onto the x-y plane. The z position should not be constrained at all. Thus, the two arbitrary points might be (X1, Y1,0) and (X2, Y2, max).*

The scan will terminate when the finger either exceeds one of the bounds, or returns to its initial position. This second termination condition is useful if the host is interested in producing a contour map of the object. It could do this by requesting a series of scans, using cross-section planes parallel to the x-y plane, but at varying z values. In this case we would like the finger to

---

* In addition to this constraint, there is an implicit maximum search volume given by the dimensions of the device.

completely circumscribe the object, continuing until it returns to its starting point.

A problem which has not yet been mentioned is that of keeping in contact with the surface of an object. It turns out that in most situations, this is relatively simple. The method requires three kinds of information.

As described earlier, the finger has a number of distinct faces. The present structure of the positioning device does not allow for rotation or re-orientation of any kind. Hence, except for possible translation, these faces are fixed. Their equations, as well as those of the planes perpendicular to them, are predefined as constants in the MCP program.

Second, we have the equation of the cross-sectioning plane. All motion of the finger is to be restricted to that plane. By intersecting this plane with either the plane of a face of the plane perpendicular to a face, we can calculate a line of motion. This can then be fed to the absolute move routine to effect the movement.

Finally, there is the data from the Tactile Sensing Processor. This indicates whether each face is below range, within range, or above range. Typically, there will be only one face which is within range. This is labelled the "active face," because it is the one which is in contact with the surface. There are exceptions, and we will see

shortly how we can account for them.

The objective in keeping in contact with a surface is to keep the active face within range. Recalling that by definition of the command, the active face is initially within range, we have the following cases:

(1) Active face is within range;
(2) Active face is below range;
(3) Active face is above range; and
(4) A second face comes within or above range.

In case (1), the finger is in contact with the surface. Our best estimate of the shape of the object at this point is a plane parallel to the active face. Calculate the line of motion (if it has not been calculated already) as the intersection between the active face and the cross-sectioning plane. Send the current position to the host, and take a step.

In cases (2) and (3), the finger either has lost contact, or is pressing the surface too hard. Calculate a line of motion as the intersection between the cross-sectioning plane and the plane perpendicular to the active face. Then take a step along it away from or toward the finger's center, respectively. Do not send this step vector to the host, because it is not part of the surface contour.

Case (4) could result from several different situations. Take the scenario in which the finger hit a

concave corner. In this case, the appropriate action is to make the new face the active face, and then act according to its status.

Another scenario in which case (4) could occur involves reaching either a convex corner, or a point at which the surface curves away from the currently active face. Again, the appropriate action is to declare the new face as the active face, and act according to its status.

There are a number of other situations in which a second face could come within or above range. The appropriate action is not always the same as above. In fact, one could imagine situations in which a third and perhaps a fourth face must be considered. Though these cases have not yet been adequately resolved, we do not expect them to be overly troublesome.

Chapter 3: The Implemented Software

We noted earlier that although the software was designed for the finger, it was eventually implemented for the pad sensor. The most notable difference between design and implementation was the fact that in the end, we only used one microprocessor. All those commands which required multiple face sensing -- trace contour, follow edge, etc. -- were eliminated because the pad sensor in fact has only one face. It happened that these commands coincided with the ones which required real time feedback. Therefore, the requirements of the tactile data acquisition software became almost trivial, and could be handled easily and much more simply by the Motor Control Processor.

3.1 Environmental Details

The microprocessor software is written in Z80 assembly language. It resides on the PDP-11/60, which runs under the RSX-11M operating system. We use a primitive Z80 assembler, written in C, which produces Intel hex-format object code. This we download to the microprocessor via the 1200 baud serial line which connects the two systems. As it turned

out, 1200 baud was as fast as the 11/60 could reliably receive and store data.

The microprocessor system is made up of a California Computer Systems S-100 bus and mainframe, 8K of RAM, and a Cromemco Single Card Computer (SCC) with 1K RAM and room for 8K of PROM, 1K of which is taken up by a modified form of Cromemco's power-on monitor. The SCC has five timers, three parallel ports (input/output), and a serial port. Since the A/D converter built into the pad sensor produced CMOS output levels, we decided to temporarily add our own converter, a Cromemco D+7A board.

In the following sections we give a complete description of the software as it currently stands.

3.2 Command Format and Interpretation

The command language was to be a permanent part of the software. It would be used initially by a human user to control the pad sensor's movement and data acquisition. Eventually, however, it would become the Experimental Sensory Processor's way of driving its tactile branch.

Thus we had three goals in mind. First, the command language should be versatile. It should be able to handle the commands described in the previous chapter as well as the simple placement and data acquisition commands we needed for the pad sensor experiments. Second, it should be

concise enough, and easy enough to interpret, to be used for interprocessor communication. Finally, it had to be legible, so that the user could issue commands from his keyboard.

We settled on a syntax with mnemonic, single character commands, optionally preceded by an ascii-coded positive or negative integer which defaults to +1 if omitted, and optionally followed by any special arguments required by the command. The preceding integer is decoded by the parser. It generally refers to the multiplicity, though its interpretation is up to the individual command routines. The trailing arguments are parsed and interpreted completely by the individual command routines.

Commands may be strung together to form a command sequence. Execution will not begin until a carriage return is received. The sequence is, of course, stored in a buffer until execution is complete. A key advantage to this is that it makes loops possible. In the syntax, a subsequence may be grouped by parentheses, which in turn may optionally be preceded by a multiplicity M. The entire subsequence will be repeated M times. Subsequences may be nested to any reasonable depth.

There is one more rather important feature. While the command sequence is incomplete, the Motor Control Processor completely disables interrupts. Since the motors are driven by periodic timer interrupts, all movement must stop.

Similarly, characters coming from the serial line during command execution are ignored. This generally does not matter, because execution will have terminated before a new command sequence arrives. However, should it become necessary for the host computer (or user) to abort execution, it (he) may send an ESCape character. This causes a non-local subroutine return to the command sequence input routine, which immediately disables interrupts.

The following is a list of the commands currently available.

H	Home -- return to inner, upper left corner, and reset the current position to (0,0,0).
nX	Move n steps in the X direction (n may be positive or negative, and defaults to +1 if omitted).
nY	Move n steps in the Y direction.
nZ	Move n steps in the Z direction.
@x,y,z	Move to absolute position (x,y,z).
n(	Begin nest.
)	End nest.
=	Return current position as x,y,z coordinates, ascii-coded decimal values separated by commas.
Q	Quit the program -- return to power-on monitor.
1S	Take a snapshot of the sensor, store data in memory, increment frame count.
-1S	Take as many snapshots as possible until the completion of the current motor step.
0S	Clear the frame memory.
G	Send the contents of the frame memory to the host, beginning with the frame count. All data is in ascii-coded hexadecimal. Then clear the frame memory.
space	Null operation.

These commands are obviously very simple. However, they can be very powerful when grouped together. For example, the sequence

```
@100,100,100 50( 3( 20X 20Z S -20Z) 20Y -60X) G
```

takes 150 snapshots, in a 50 by 3 grid, beginning at (100,100,100), then sends all the collected data to the host computer. Since optical limit switches prevent the motors from moving past the ends of travel, one could find the maximum limits in all directions by issuing

@10000,10000,10000 =

(the actual range is roughly 1200 steps per axis). This would move the sensor to the corner opposite the home position and report the actual coordinates.

This list will eventually be enhanced to include the commands described in the previous chapter. We expect to be able to continue to denote each command with one mnemonic character.

3.3 Motor Control

It is not surprising that the most complicated task performed by the Motor Control Processor is, in fact, motor control. The complexity arises for two reasons. First, it is intended to be a permanent part of the MCP software, and is therefore very general in design. Second and most important, the step service routines effectively and completely insulate the higher level command execution processes from the hardware.

At the top level, an individual command routine uses

the step services in the following fashion:

```
Set the direction components in LINE
Call SCFILL to fill the step control table
Do until termination-condition:
    Call STEP to initiate a step when ready
    Call NEWPOS to update current position
    Call NEXTPO to prepare the next step
End
```

Note that it does not concern itself with timing in any way, nor does it have to take into account the physical limits of the device. The STEP routine guarantees a minimum pulse width (maximum step rate), and even modifies the step request if such an action would drive a motor past its end of travel.

Also note that the routine must actively request that a step be taken. If, for some reason, the evaluation of the termination condition is very time consuming, the motors will simply run slower. This has another advantage. Should the program be damaged by an unusually high incidence of cosmic rays, the motors will not go out of control. They will simply stop, because nothing is calling the STEP routine.

Before we take a closer look at these routines, we must discuss the data structures involved. The first one that was mentioned is LINE. It takes three numbers to define the direction of a line in 3-space: delta-x, delta-y, and delta-z. These are the line's direction components. Simply put, when we take delta-x steps in the x direction, we must

also take delta-y steps in the y direction, and delta-z steps in the z direction. Within the MCP, these values are stored and manipulated as unrestricted 16-bit integers. However, should it later become necessary to compare line directions, these may have to be restricted to relatively prime integers. LINE is a three word array which defines the desired path to the step routines.*

A commonly accepted canonical form for these values is a list of direction cosines. This requires that the values be real numbers, and that the sum of their squares equal unity. Fortunately, we have not found this form necessary.

The second data structure is the Step Control Table (SCTAB). This 15-byte table is basic to the operation of the step service routines. Following is a layout of its contents.

```
SCTAB+ 0: (byte)  Next port image
       1: (byte)  Port image skeleton (direction bits)
       2: (word)  Max direction component
       4: (word)  Mid direction component
       6: (word)  Min direction component
       8: (word)  Mid accumulating counter
      10: (word)  Min accumulating counter
      12: (byte)  Max direction's motor pulse and power bits
      13: (byte)  Mid direction's motor pulse and power bits
      14: (byte)  Min direction's motor pulse and power bits
```

Let us digress a moment before we explain SCTAB. Instructions are passed to the stepper motors via an 8-bit

---

* The Z80, of course, does not really have any distinct concept of a "word." However, being an old PDP-11 man, I always have and always will refer to a 2-byte quantity as a word.

output port, which looks like this:

bit 7	6	5	4	3	2	1	0
Z dir	Z step	Y-Z power	Y dir	Y step	X power	X dir	X step

The three direction bits indicate which direction the corresponding motor is to move. One implies the negative direction, zero implies the positive. The step bits, when pulsed, cause their corresponding motors to take a step in the indicated direction. Due to a low-pass filter which is applied to these bits for noise immunization purposes, there is a minimum pulse width. The MCP uses a separate timer for this, as will be described later.

Finally, the power bits, when on, cause drive power to be applied to the corresponding motors. For now, the reader need only understand that a motor must have power in order to operate.

Now we should be able to make sense out of the Step Control Table. The first item, the "next port image" is exactly that -- the 8-bit quantity that is to be sent by the STEP subroutine to the motor drive output port at the next opportunity. It is very important to note that this value is, in general, calculated concurrently with the previous step, by a call to NEXTPO.

The second item, the "port image skeleton," contains the three direction bits. These bits are applied with every

step. The SCFILL routine sets them according to the signs of the three direction components in LINE, and they do not change again until a new line is chosen.

The next three items, the Max, Mid and Min direction components, are actually the magnitudes of the numbers that appeared in the LINE array, but in sorted order. These are used in conjunction with the Mid and Min accumulating counters to determine which motors to step at the next timing interval.

Finally, the mapping from the sorted order to the x-y-z order is given by the last three items. Each of these bytes has exactly two bits set, corresponding to the appropriate motor's step and power bits.

The NEXTPO routine first decides which motors are to be stepped, and then adds together the corresponding mapping bytes, along with the direction bits from the skeleton. The resulting value is the next motor port image.

Let us now return to the high level control loop given at the beginning of this section. First of all, note that the values passed in the LINE array indicate a direction only. They do not completely describe a line segment in 3-space. It is assumed that the line of motion will begin at the current position, and the control loop is responsible for knowing when to stop.

Once the LINE table is set, SCFILL is called to fill

the Step Control Table. All values are calculated independent of the previous contents. The NEXTPO routine is then called automatically to use the new table to compute the first port image and place it in the zeroth location.

Since a step is never taken unless specifically requested by the control loop, it is perfectly reasonable to completely change direction at any time by simply changing LINE and calling SCFILL, before calling STEP again. One need not be concerned with the timing considerations.

Within the control loop itself, the first action is a call to the STEP routine. This routine waits, if necessary, for the previous step to complete. Then it calls CHECK to check the optical end-of-travel limit switches and, if necessary, modify the candidate port image. Finally, the routine outputs the image to the motor port and returns to the calling control loop.

Internally, one of the five on-board timers is also set to cause an interrupt after a time equal to half the minimum step pulse width has elapsed. The routine which handles that interrupt will clear the motor step bits and set the timer to interrupt again after another equal interval. At that point, an entire step has completed. The STEP routine, if it is waiting, is allowed to proceed with another step. In this way, something like an open ended square wave is generated on the motor pulse bits.

This brings us to the other subroutine calls in the main control loop. During the timing delays, the CPU is free to do quite a substantial amount of processing. Recall that the STEP routine has the power to modify the candidate port image. This modified image is returned to the control loop, where it is passed again to the NEWPOS routine. NEWPOS, based on the direction and step bits which were actually sent, updates the current coordinate counters.

The calculation of the next port image is then accomplished by a call to NEXTPO, which proceeds as follows.

1. Begin with the motor port skeleton, which defines the direction bits.
2. Add in the Max direction's pulse and power bits. That motor is to move at the maximum rate, and will therefore always take a step.
3. Add the Mid direction component to the Mid accumulating counter, and take the result modulo the Max direction component. If there was an overflow, we want to step the Mid motor. Add in its pulse and power bits.
4. Repeat step 3 for the Min direction.

The resulting value is placed in the first byte of the Step Control Table. An example of this algorithm in operation was given in chapter 2.

There is one final item to discuss. Conceptually, a stepper motor has a series of magnetic coils arranged in a circle around an iron core. As steps are taken, each coil in succession is energized, drawing the core around the circle. During normal operation, a given coil is only

energized for a brief period before its successor takes over. However, when the motor is standing still, one coil is energized continuously for a long period of time. It can generate quite a bit of heat -- enough, perhaps, to burn itself out.*

To solve this problem we implemented the following scheme. Every time a motor is stepped, its power is automatically turned on. At the same time, its corresponding usage counter is reset to some constant. Periodically, another of the on-board timers interrupts the processor to decrement all the usage counters. When any one reaches zero, the corresponding power bit is turned off.

The effect of this is to power down any motor that has not been stepped in the last two seconds. The action is so completely transparent to the higher level control software that we refer to it as the "burnout protection demon."

3.4 Tactile Data Acquisition

Due to its temporary status, the tactile data acquisition is perhaps the least important part of the software. As soon as the finger arrives, these routines will be removed from the Motor Control Processor and rewritten completely for the Tactile Sensing Processor,

---

* I don't know whether motors would actually burn out, but when I found I could fry eggs on them, I did not want to take chances.

according to the plans given in chapter 2. Therefore, as might be expected, the current code is far from general. It is entirely driven by the S and G commands described earlier. Nothing happens asynchronously.

The entire unused portion of the MCP's memory board is used as a buffer for tactile data. Upon MCP initialization, the frame count is reset to zero. Then, each time a snapshot is requested, the data record is placed in the next position in the buffer, and the frame count is incremented. When the readout is requested (via the G command), the program simply types it all out, one line per record, beginning with a line consisting solely of the frame count. The information is transmitted in ascii coded hexadecimal, as an optimization of both transmission time and coding time.

Chapter 4: Experiments and Results

In this chapter we will discuss the experiments which were actually performed using the pad sensor. We will consider the methods, the goals, the problems, and the results. When possible and appropriate, we will refer to figures which illustrate the results.

4.1 Calibration

The pad sensor consists of an 8 x 8 array of sensitive sites whose analog output values are fed into an analog multiplexer, and finally into an analog to digital converter. All this circuitry is part of the sensing device. Unfortunately, since the A/D converter emits CMOS voltage levels, and our parallel ports use TTL inputs, we had to bypass the internal A/D and use our own. This resolved the incompatibility, but gave vent to another problem. The pressure signals coming out of the multiplexer ranged roughly from +2.0 to +2.5 volts, and our A/D converter expected a range of -2.5 to +2.5. As a result, the digital pressure readings never went below about 235, out of a maximum 255.

In other words, the fact that we can exhibit only a

little over four bits of precision is not a reflection on the device, but on the interface. With the right interface, we would estimate upwards of six bits of valid data.

Each of the 64 pressure sensitive sites puts out a slightly different range of voltage levels. They therefore required individual calibration. The most straightforward way of doing this is to press the sensor down hard on a flat surface, take a snapshot, release the sensor entirely, and take another snapshot. This yields a matrix of minimum and maximum pressure values, to which all subsequent data would be scaled in a linear transformation.

Of course, nothing is ever so simple. Each pressure sensitive site requires roughly 1.3 pounds of pressure to completely depress it. Multiplying that by 64 sites, we find that we need over 80 pounds of pressure to acquire the maximum readings. Our Z-axis motor is not capable of this.

The solution was to depress each site individually, and then combine the data into a single matrix of maximum pressure values. Fortunately, the Motor Control Processor's command language was flexible and powerful enough to do this painlessly in one command sequence, with two loops for X and Y positioning.

Once the minima and maxima were obtained, it was a simple matter to map all input data into a uniform range of 0 - 255. It is worth mentioning here that throughout the

entire testing period, these ranges never changed more than one unit. In addition, we never had any problem with spurious data being generated where there was no contact. Those points always mapped to zero. We were quite impressed with the robustness of the pad sensor.

4.2 Static Tactile Image Analysis

4.2.1 Single Image

The obvious first step in analyzing tactile images is to lay the sensor down on a known object, take a snapshot, and see whether it is recognizable. This we did, and the results are depicted in fig. 1.

In fig. 1f we used a one inch square, set off-center, but oriented orthogonally with the sensor's grid axes. There is no question as to the identity of that object. A simple threshold operation would clearly distinguish it from the background.

Fig. 1e and fig. 1d show the same square rotated counterclockwise 30 degrees and 45 degrees, respectively. Fig. 1c shows an equilateral triangle, point downward, and fig. 1b depicts the same triangle rotated clockwise about 75 degrees. Notice how some pixels are much lighter than others in the images with non-orthogonal edges. This phenomenon arises when the object covers less than half the area of a site. Since the site is conical in shape, the

edge must be pressing on the wall of the cone. It cannot depress the cone as far as it could if it were pressing on the apex.

In theory, it should be possible in some cases to determine exactly how much of the cone is actually covered by the object. However, we must assume the following: 1) that the object surface, particularly the edge in question, is smooth, 2) that the object surface is in a plane parallel to that of the pad sensor, 3) that the individual sites on the sensor are in fact conical, with bases that meet the bases of their neighbors, and 4) that we know how to calculate the actual depression as a function of output pressure value.

Unfortunately, neither of the last two assumptions are valid in our case. The cones are actually cut off before they reach the apex,* and we do not have the data to perform the depression calculation.

Finally, fig. 1a shows a one inch diameter circle. Notice that it appears to be identical to the square in fig. 1c. This is a question of resolution. Clearly, if the spatial resolution were doubled or quadrupled, the distinction would be obvious.

---

* My office-mate tells me that the technical term for this shape is "frustum."

4.2.2 Spatial Resolution

How do variations in sensor resolution effect the image? The simplest way to tackle this question is to vary the size of the features on the test objects. We used a set of disks with raised concentric circles projecting from them in relief. The variations consisted of two amplitudes and three frequencies, totalling six disks.

Fig. 2 shows the images obtained. As might be expected, those disks in which the spacing between the circles approach the spacing between the sensitive sites (figs. 2a and 2d) are clear. As the frequency increases, the shape becomes less obvious, until it is completely unintelligible at the highest frequency.

The effect of amplitude is also fairly predictable. At low amplitude, the circles are wider, and therefore more sites are in contact with the surface. This can be seen most clearly (again) in figs. 2a and 2d. Also, the inner circle is more distinct in fig. 2e than in fig. 2b. This is because at the lower amplitude, the depth of a trough is considerably less than the height of a conical site, and therefore some trough sites come in contact with the surface.

Theoretically, it should be possible to compare pressure values and determine where the troughs and crests occur. However, here we run into the limitation in our 3-D

positioning device which we alluded to in the Calibration section. The Z-axis motor, which supplies the normal force, is a bit too weak for this pad sensor. Each sensitive site requires a certain amount of force to depress it, and the motor must be able to exert the sum of these forces in order to obtain a reliable reading. Therefore, as more sites contact the surface, each one receives less pressure. Furthermore, if the surface is not uniform, neither are the reductions in pressure.

4.2.3 Multiple Images

How can we improve the spatial resolution with the equipment available to us? One simple way to double the number of data points on each dimension is to take a reading at each of the four corners of a small square, whose sides are half the length of the distance between sites. This we did, using the same six disks, and the results are visible in fig. 3.

The images are slightly clearer, but not as much as we had hoped. Again, the disappointment is indirectly caused by the deficient Z-axis motor. When taking a snapshot, we try to depress the sensitive cones as much as possible, since we are not capable of depressing any of them completely. To do this, we simply instruct the Motor Control Processor to lower the Z-axis motor until it won't go any further.

This works quite well in general. However, consider the following hypothetical case. Suppose the test object is a single sine wave and the sensor is a single cone. First, we lower the cone onto the crest of the wave as far as it will go, and take a snapshot. Then we move the cone to the trough and repeat the operation. The two images look identical! In both cases, the cone was depressed as far as it would go, and it is in fact the cone depression which determines the image. This, we believe, is the root of the multiple image problem.*

The solution, of course, is to strengthen the Z-axis motor. Then, instead of simply lowering the sensor until it stops, we would lower it to a consistent Z-coordinate. The resulting set of images would be much clearer.

4.2.4 Large Objects

Can we examine objects which are much larger than the sensor? For this experiment we used a flat surface about 12 inches long and three inches wide -- slightly wider than the sensor pad itself. A set of eight grooves were cut into this surface in order to form a pattern of diverging lines (see fig. 4a). By taking a series of snapshots at successive lengthwise positions, we should be able to reconstruct the entire image, in spite of the fact that it

---

* Or, "Aye, there's the rub!"

is much longer than the sensor.

The Motor Control Processor's command language again made this a simple task. We took fifty images, stepping about five millimeters between each. The reconstruction, shown in fig. 4b, was accomplished by superimposing the images in the appropriate positions relative to each other. As before, when the distance between features approaches the distance between sensitive sites, the pattern becomes clearer.

Can we use our multiple image trick to improve the resolution? We repeated the same procedure, except that this time we took three snapshots, four millimeters apart widthwise, for each of the fifty steps lengthwise. The reconstruction, fig. 4c, shows the angled edges much more clearly at lower frequencies than does fig. 4b. At higher frequencies, however, both reconstructions are equally unintelligible. Once again, we blame the failure on the Z-axis motor, and our method of maximizing pressure.

4.2.5 Small Angle Measurement

When a robot hand grasps an object, does it have a good grip? Very often, a "good grip" is one in which the flat surfaces of the object are wholly in contact with the flat faces of the fingers. The question can then be answered very simply by measuring, for each finger, the angle between these two planes.

This experiment proved to be extremely successful. Using the one inch square as our test object, we took four snapshots. In the first image we layed the pad sensor flat on the square, as usual, giving us a zero degree standard. For the three subsequent images, we lowered the left end of the table by 1.0, 1.25, and 1.5 inches respectively, producing angles of 3.3, 4.1, and 4.9 degrees.

The results are shown in table 1. For each image we arrived at a single number describing the slant. The number was calculated simply by averaging all the pressure differences between horizontally adjacent sites. In theory the ratio of the third slant value to the second should be 1.25,* and the fourth to the the second should be 1.5. This was not the case.

However, the first image, whose slant should have been zero, did exhibit a small slant value. If we take this as an error, we can produce a correction factor by dividing it by the slant value for the second frame. When that percentage is subtracted from each of the two ratios arrived at earlier, we get remarkable results. The corrected ratios differ from the expected values by less than two percent!

---

* Proof is obvious from the geometry, as long as we assume a linear relationship between depression distance and output value.

4.3 Dynamic Texture Analysis

We believe that until tactile sensors can be fabricated with extremely fine resolution, information about the texture of a surface would best be obtained by moving the sensor along the surface, and examining the changes in pressure readings, as opposed to the pressure readings themselves.

Toward this end, we tried several times to make the positioning device drag the pad sensor along different surfaces, but failed each time. The sensitive cones, because they were designed to grasp an object without allowing it to slip, were made out of high friction rubber. This, of course, directly hindered the experiment. The stepper motors were not powerful enough to pull the sensor and still maintain enough contact pressure to yield a significant reading.

In the end we performed a singularly unscientific experiment. We dismounted the pad sensor from the positioning device and dragged it by hand along a flat wooden surface, taking 100 snapshots over a period of about five seconds. This may not have been so bad, except that we neglected to measure the exact distance traversed, or anything that could directly or indirectly give us the velocity.

The analysis is interesting, though quite inconclusive.

The sensor is made up of an 8 by 8 grid of sensitive cones. Let us define a column as the series of cones lined up in the X-direction, and a row as the cones lined up in the Y-direction. Given that the sensor was dragged in the positive X-direction, we contend that there should be some aspect of the data which is consistent down a column, but different across a row. Furthermore, there should be a small but constant time delay between the features exhibited by one site and those exhibited by the next site down the column.

The motivation for this hypothesis is as follows. Picture a textured surface as a terrain of bumps and ridges. As the sensor grid passes over this terrain, the cones across a row will collect entirely unrelated data. However, those down a column will encounter the exact same bumps and ridges that were encountered by their predecessors, but a little bit later. Thus we have eight instances of eight-fold redundant data. We should be able to find some consistency somewhere.

Initially, we plotted the raw pressure data from each of the 64 cones as a function of time. Fig. 5 is a reproduction of this, with each plot placed in the same grid position as the corresponding cone. We expect to be able to look down a column and see some consistency that does not occur across a row. Unfortunately, no such consistencies were immediately obvious.

The next step was to try to home in on the changes in pressure, as opposed to the pressures themselves. However, a simple pairwise difference derivative (see fig. 6) was no more enlightening than the raw data.

Well, what about the Fourier transform? Surely the frequency domain is closer to our goal than the time domain. Unfortunately, applying this transform meant giving up our time delay information, which we needed for comparing curves.

What we really needed was some smooth measure of frequency as a function of time. A colleague* suggested the following procedure. First, take the pairwise difference derivative. Then, pass a window along the time axis. For each point in time, count the number of zero crossings in the window, and divide by the width of the window. A window n units wide would have a maximum of n zero crossings, and thus the ratio would be unity. No crossings would produce a ratio of zero. Note that the operator is valid, and produces the same range of values, independent of the window size. The only difference is in the precision.

We used a window with an odd number of points, so it could be symmetric about the point under consideration. If the distance to one margin or the other was smaller than half the window size, the window was shrunk accordingly, so

---

* Thank you, Gerry Radack.

that symmetry was maintained. We tried various window sizes in order to obtain the smoothest curve possible without losing too many features. The optimal size was about 25 units (out of 100), shown in fig. 7a. A 15 unit window is shown in fig. 7b for comparison.

There are (finally) some definitely visible similarities among the resultant curves of fig. 7a. Examine, for example, the troughs in rows 6, 7 and 8 of column 1. Notice how similar they are, and how a small, constant time delay occurs between each curve and its successor. The same phenomenon is visible in rows 1, 3, 5 and 6 of the third column, and in rows 1 and 3 of column 7.

As one looks up and down a column, there seems to be some kind of topological similarity. This is exactly what we want to find. However, identifying it mathematically is no simple task. The obvious operator to apply would be the cross correlation. This compares two graphs and produces a number describing the closeness of the match, then shifts one graph relative to the other and repeats the calculation. One correlation value is generated for each possible shift. The resulting curve shows not only how well the two graphs match, but at what time delay value the match is optimal.

Unfortunately, the results were very disappointing. No matter which pair of graphs we compared, the cross correlation never went substantially higher than zero, and the best match always occured at zero shift. Needless to

say, at least one more level of processing is called for.

4.4 Conclusions

First, it is clear that an 8 by 8 grid of pressure sensitive sites is generally not enough for pattern recognition of single static images. In most real applications, either the objects will be larger than the pad, or the features will be below the pad's resolution.

With reasonably good positioning equipment, the resolution can be significantly improved, and the size of the area under consideration considerably increased, by taking multiple images. However, this is often too time consuming, and therefore infeasible.

The straightforward solution is to increase the spatial resolution, the number of sites, or both. We have shown that when feature dimensions are comparable to resolution, shape recognition can be quite simple. This has also been demonstrated by Hillis [HILLIS-81], using a sensor recently developed at the MIT A.I. Laboratory, and of course by Briot [BRIOT-79], who used an array of binary sensors. One typical application for this might be the table sensor which was described in the introduction.

A more novel approach might be to build multijointed fingers for the robot gripper, such as the three fingered hand developed by Ken Salisbury [SALISBURY-81] at the

Stanford A.I. Laboratory. This would enable the robot to manipulate the object while transporting it, in such a way that it becomes not only feasible, but a matter of course to take multiple tactile images.

In the experiment concerning measurement of small angles, we obtained impressive results. The computed values were even more accurate than we had hoped. From this we conclude that a tactile sensor with properties similar to those of the pad sensor is eminently suited to applications involving small angle measurement, such as grip improvement.

As far as texture analysis is concerned, we believe our approach is a good one. Visually, it is apparent that we are on the right track. However, the experiment must be repeated in a much more controlled fashion, and different surfaces must be examined and compared. Then, we hope we will eventually be able to manipulate the data in such a way that we can use it to identify the surface.

Chapter 5: Further Work

As was mentioned earlier, the pad sensor was in our possession for only a short time, by no means long enough for exhaustive experimentation. In fact, many of the more interesting ideas occured to us after the sensor was returned, when we began to analyze the data.

It should be possible to calculate the coefficient of friction between various surfaces and the rubber face of the sensor. First, one must know the force as a function of digital output for each sensitive site, as well as for the strain gauges on the metal posts. Then, one would drag the sensor along the surface in question, and take force measurements. The normal force N is simply the sum of the forces on all the sites, and the frictional force F is derived from the horizontal forces given by the strain gauges. By plugging these numbers into the equation $F = uN$ one can calculate u, the coefficient of friction. This might be usable as a distinguishing characteristic between surfaces.

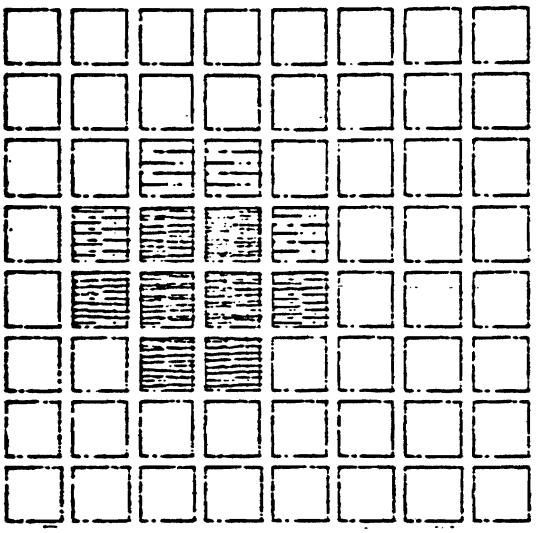
It might also be useful to measure granularity. This could be done simply by placing the sensor onto the surface

and counting the number of sensitive sites which exhibit significant pressure. Of course, the grains in the test surfaces must be comparable in size to the resolution of the sensor.

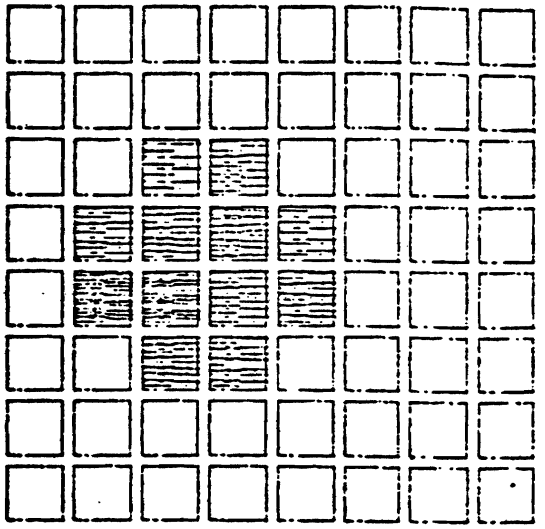
Certainly the dynamic texture analysis tests should be repeated and extended. Once that data has been hashed out, it should be possible to identify surfaces based on pressure response to friction.

Finally, there are two aspects of tactile sensing which we have not experimented with because they are better suited to the finger than the pad sensor. First, the finger should be capable of poking a surface and comparing predicted pressure with actual pressure in order to measure of surface resilience. Second, there is the whole question of tracing cross sections and producing, essentially, a 3-D description of the contour of an object.

Thus we have shape based on both static images and contour descriptions, granularity, coefficient of friction, and surface resilience and texture. These features, when they are better understood, should be incorporated as distinguishing characteristics into the Experimental Sensory Processor.

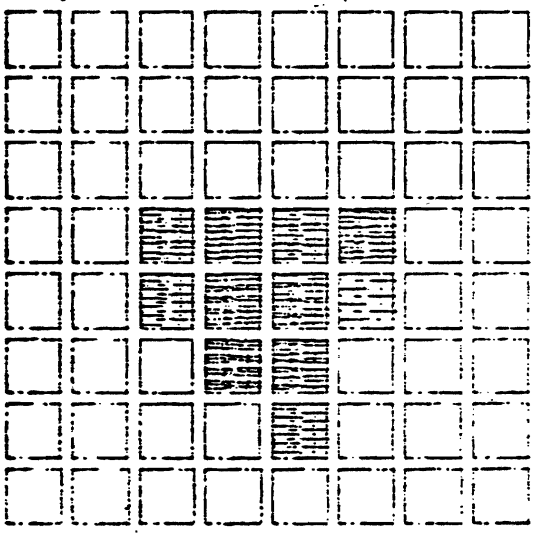


a) 1" diameter circle



d) 1" square rotated 45°

b) 1.5" triangle



e) 1" square rotated 30°

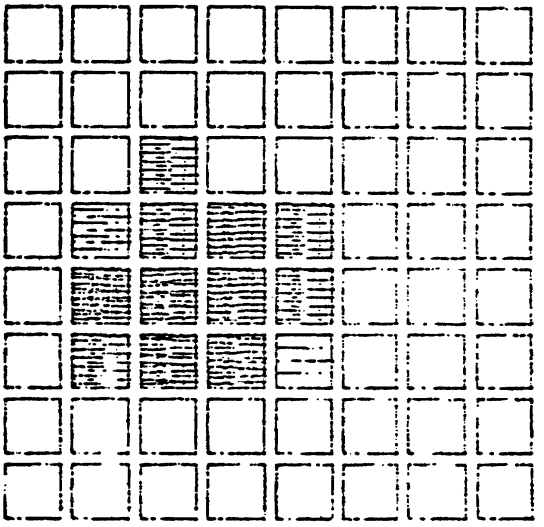
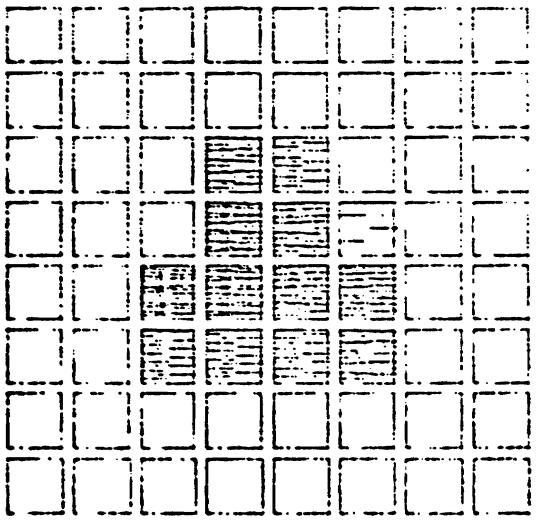
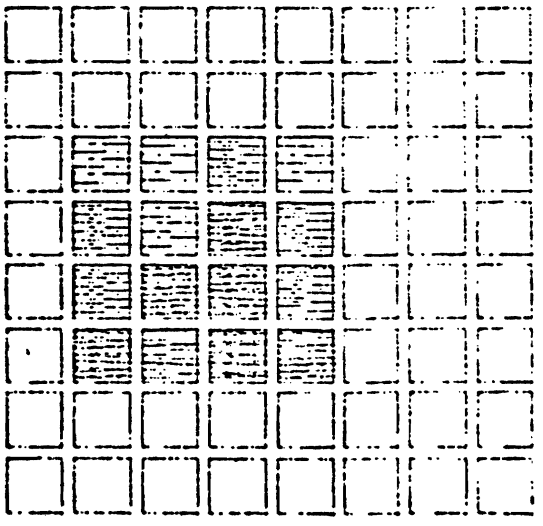


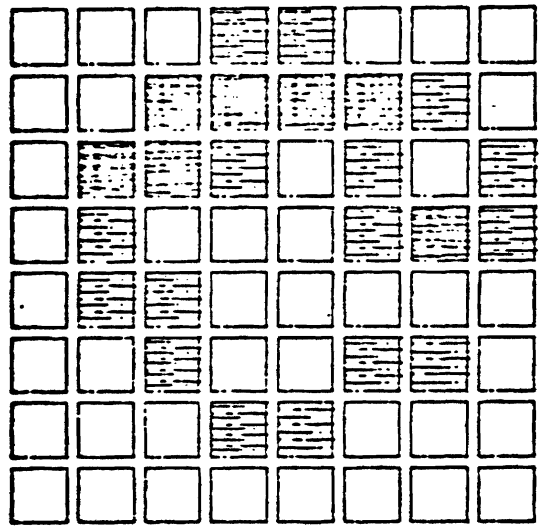
Fig. 1. Single Image Shape Recognition

c) 1.5" triangle rotated 75°

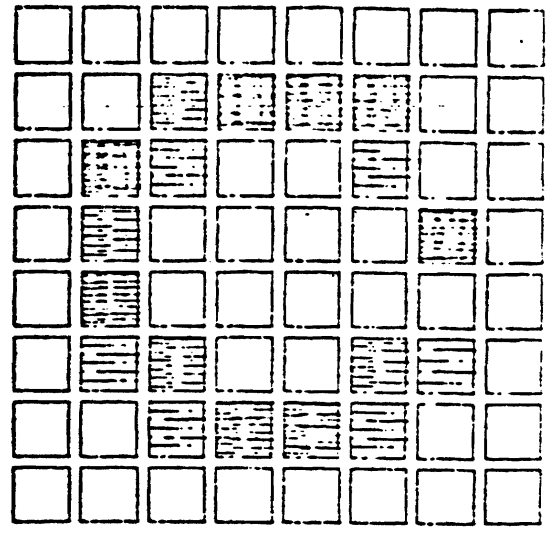


f) 1" square



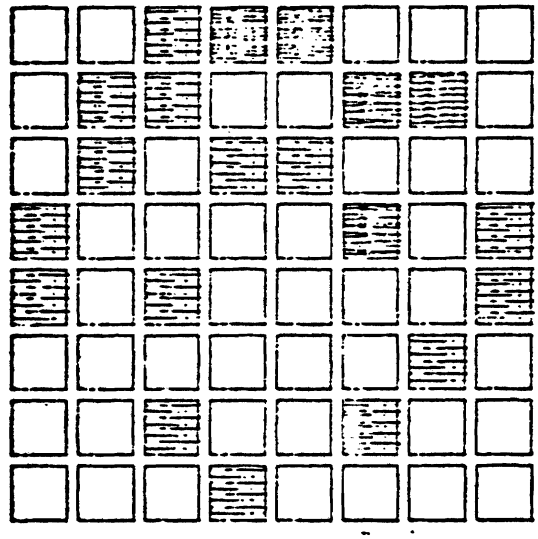


a) One circle, low amplitude

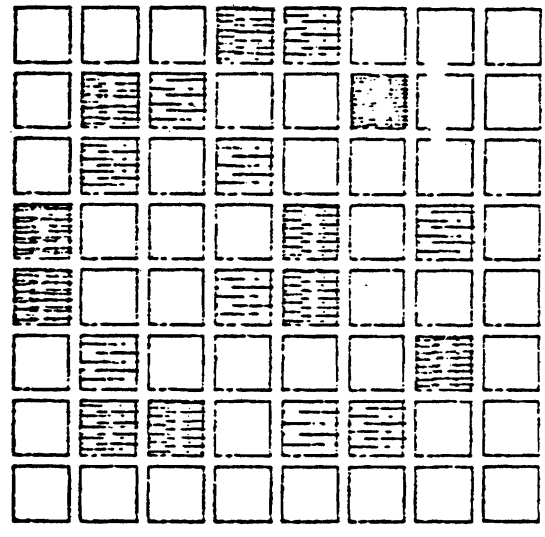


d) One circle, high amplitude

Fig. 2. Single Image Recognition Varying Frequency and Amplitude

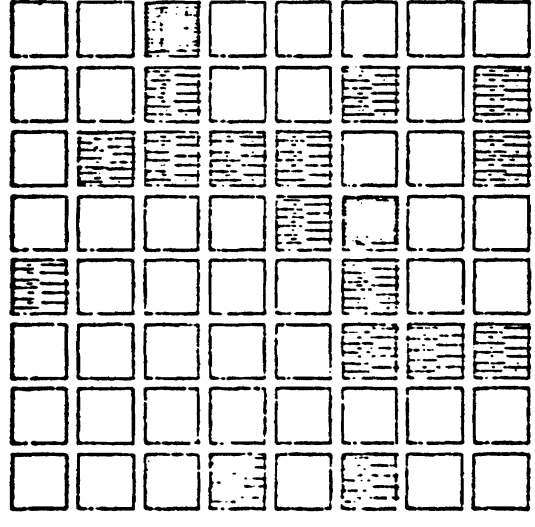


b) Two circles, low amplitude

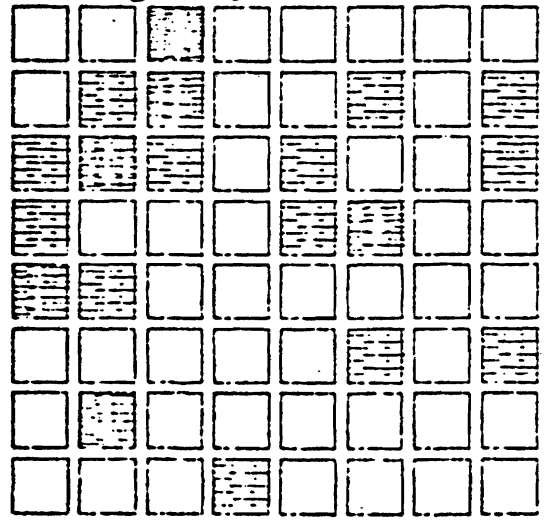


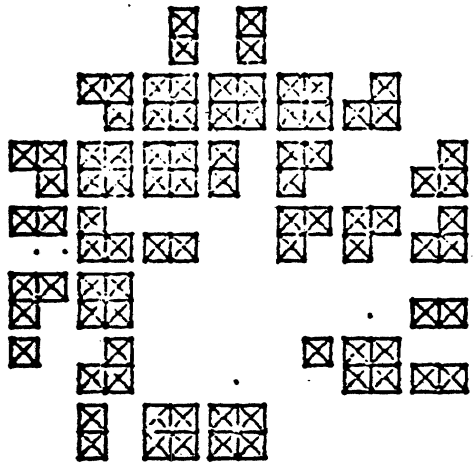
e) Two circles, high amplitude

c) Five circles, low amplitude

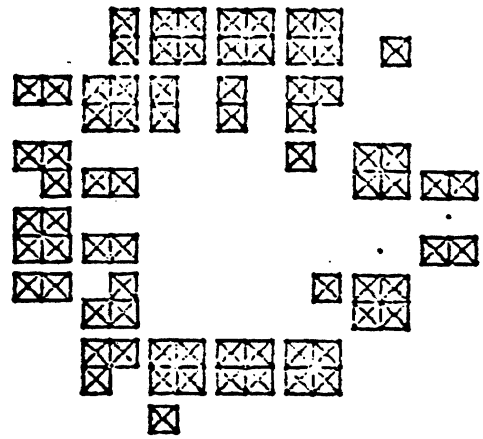


f) Five circles, high amplitude



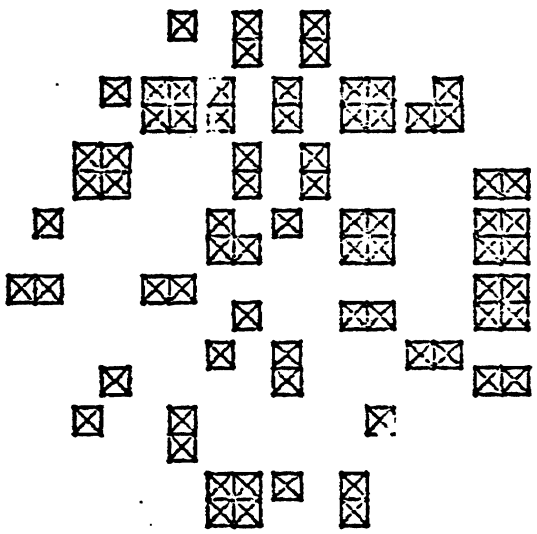


a) One circle, low amplitude

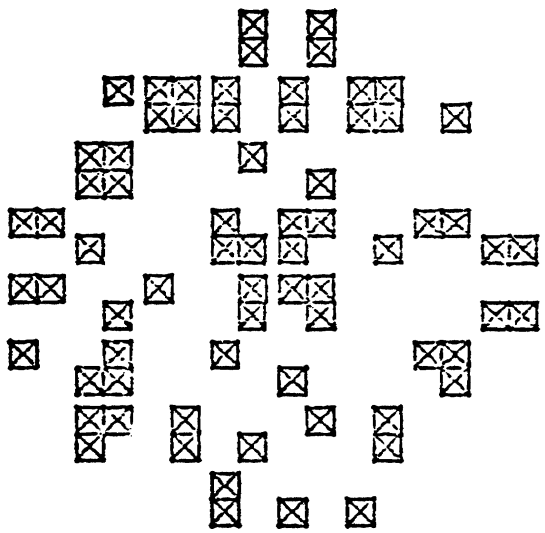


d) One circle, high amplitude

Fig. 3. Multiple Image Recognition
Varying Frequency and Amplitude

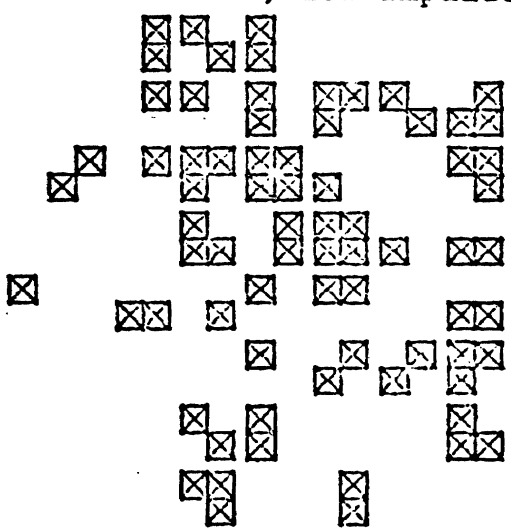


b) Two circles, low amplitude

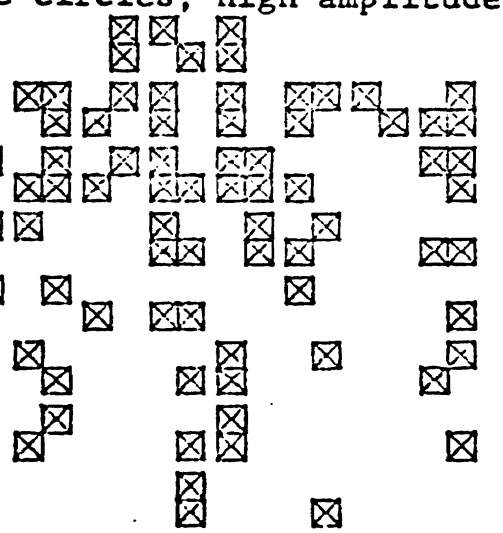


e) Two circles, high amplitude

c) Five circles, low amplitude



f) Five circles, high amplitude



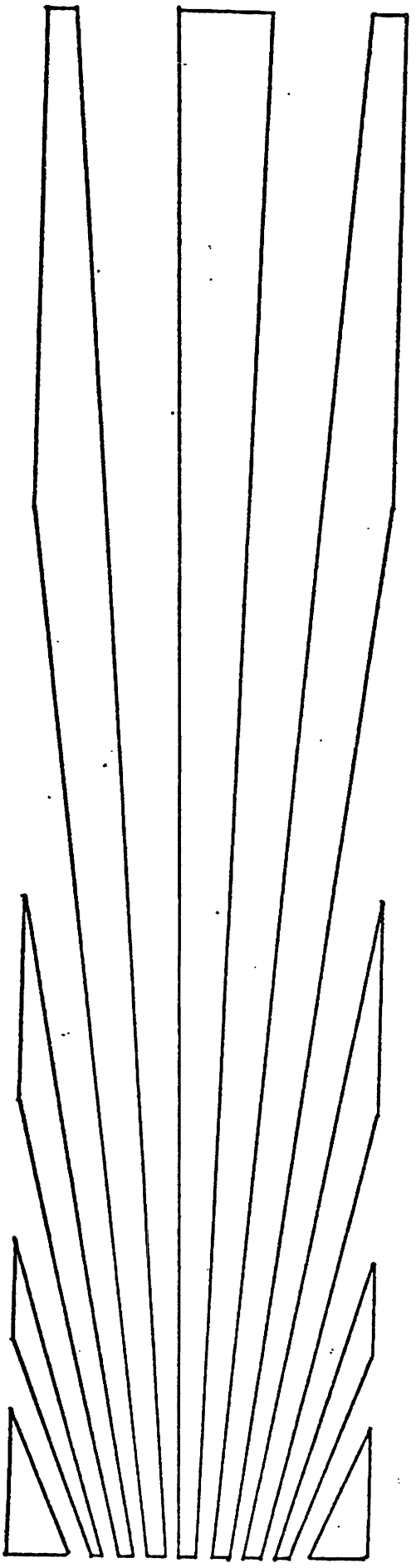


Fig. 4a.
Drawing of the
Large Test Object

Fig. 4b
Using 50 successive frames

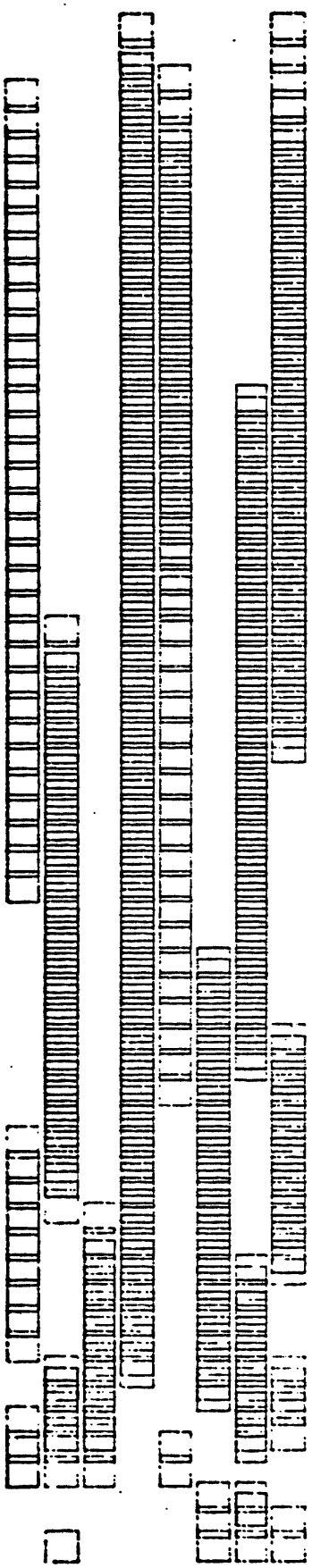


Fig. 4c
Using 3 x 50 matrix of frames

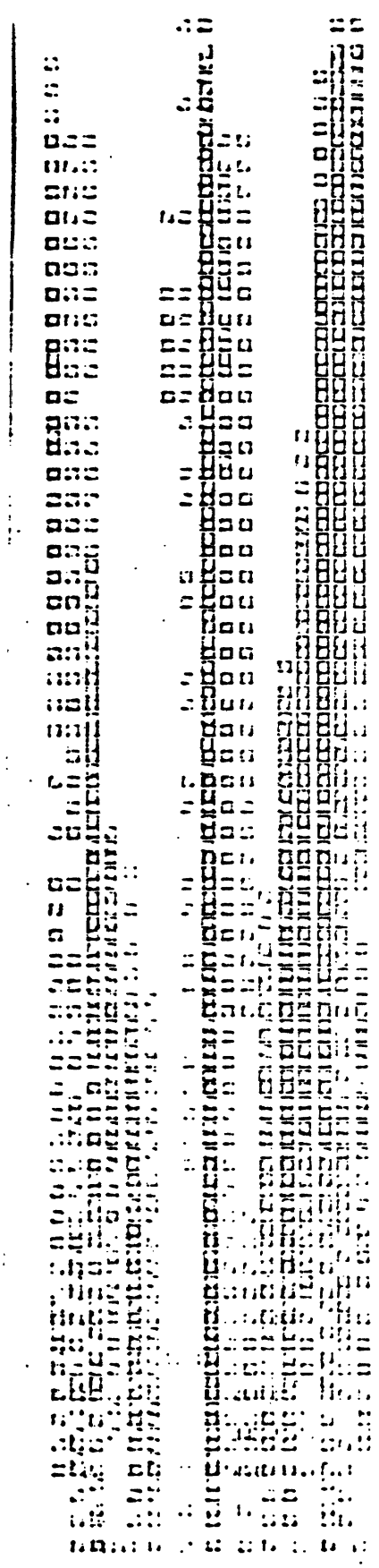


TABLE 1 -- Measurement of Small Angles

Table Slant	Data			Horiz. Difference		Avg. Diff.	Ratio*
0"	45	64	80	19	16	12.625	
	42	48	64	6	16		
	48	60	75	12	15		
	34	56	51	22	-5		
1"	15	64	160	49	96	78	1.00
	28	80	192	52	112		
	16	75	195	59	120		
	17	85	153	68	68		
1.25"		48	160		112	114	1.23
		48	192		144		
		60	180		120		
		56	136		80		
1.5"		48	160		112	142	1.53
		48	240		192		
		60	225		165		
		71	170		99		

* Ratio is calculated as the vertical average divided by the vertical average at 1" slant, multiplied by one minus the ratio of the 1" slant to the 0" slant. The closer this value is to the table slant, the better the results. As the reader can see, the results are exceedingly good.

Fig. 5. Raw pressure data as a function of time. Each plot is in the same grid position as the corresponding cone. Sensor was dragged toward +X, with Row 1 in the lead.

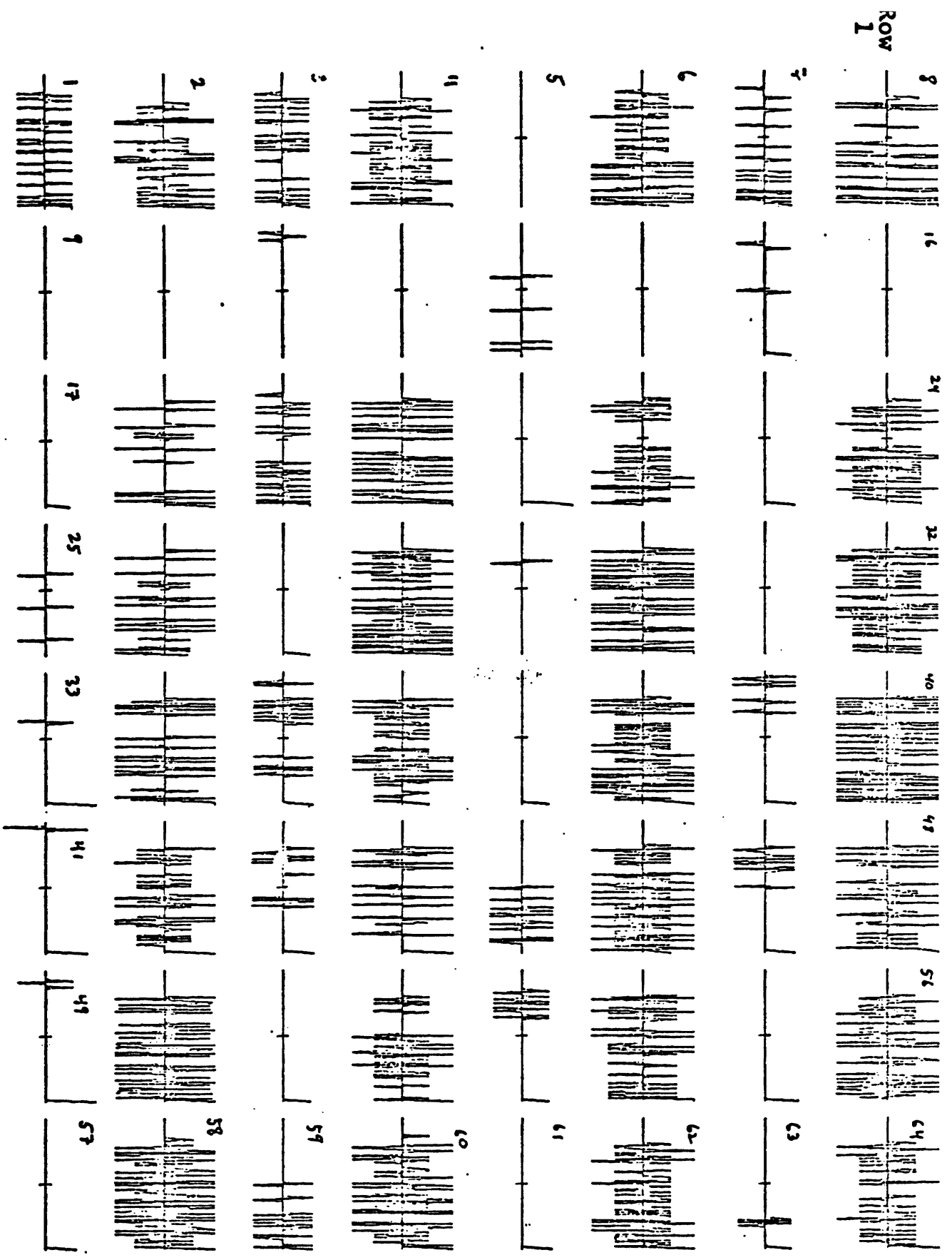


Fig. 6. Pairwise Difference Derivative

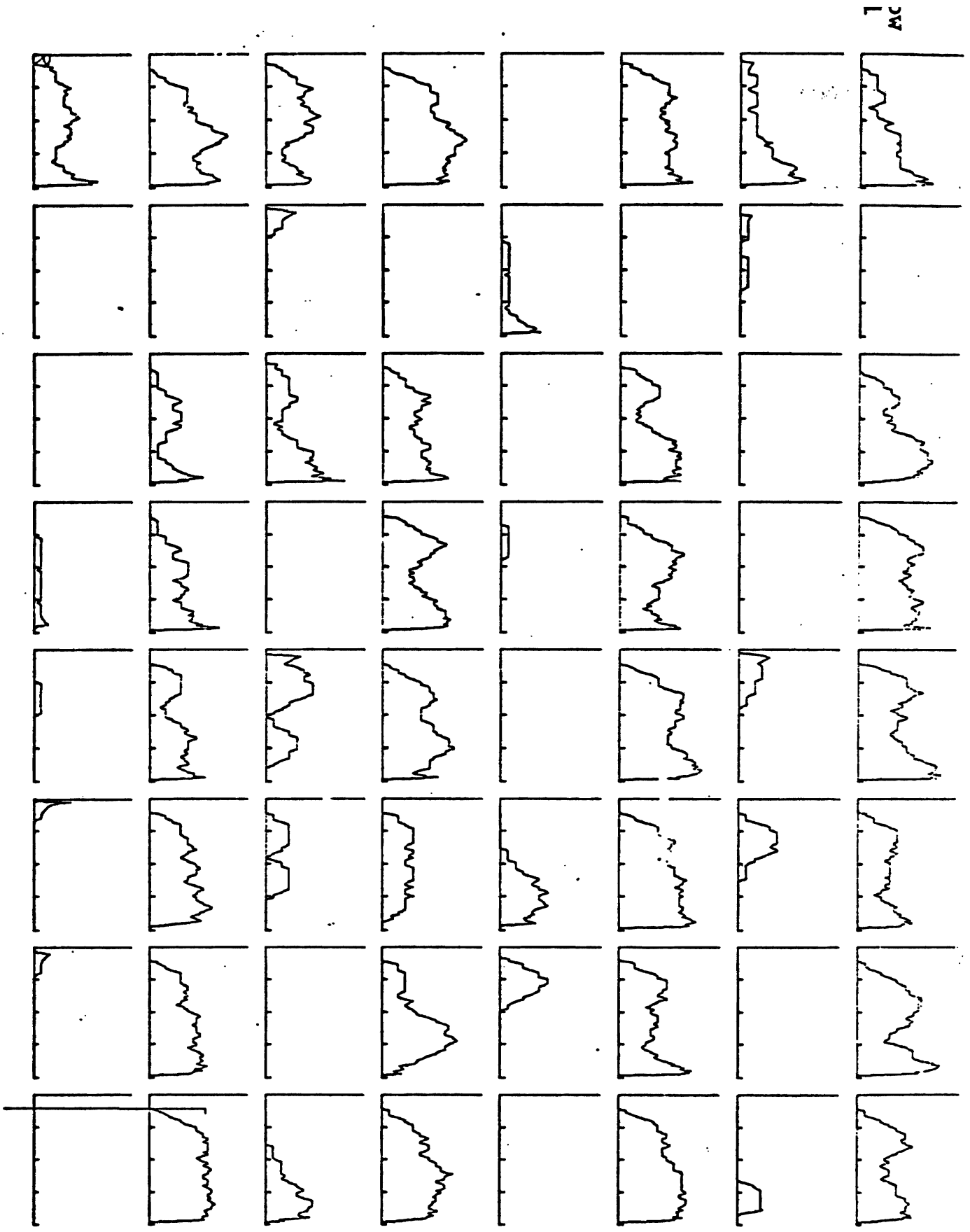


Fig. 7a. Frequency as a Function of Time
Window = 25 Units

Fig. 7b. Frequency as a Function of Time
Window = 15 Units

References

[BOYKIN-80] Boykin, W. H., and Diaz, Gary., "The Application of Robotic Sensors -- a Survey and Assessment," ASME Century 2 Conference, August 12-15, 1980.

[BRIOT-79] Briot, Maurice, "Utilization of an 'Artificial Skin' Sensor for the Identification of Solid Objects," Proc. of 9th International Symposium on Industrial Robots, Washington, D.C., March 12-15, 1979.

[BROWN-80] Brown, David J., "Computer Architecture for Object Recognition and Sensing," Master's Thesis, Department of Computer and Information Science, University of Pennsylvania, December, 1980.

[DANE-81] Dane, Clayton, Forthcoming PhD. Dissertation, Department of Computer and Information Science, University of Pennsylvania, 1981.

[HILL-73] Hill, John W., and Sword, Antony J., "Touch Sensors and Control," in Remotely Manned Systems -- Exploration and Operation in Space, ed. by Ewald Heer, California Institute of Technology Press, Pasadena, California, 1973.

[HILLIS-81] Hillis, William Daniel, "Active Touch Sensing," Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 629, April, 1981.

[IVANCEVIC-74] Ivancevic, Nebojsa S., "Stereometric Pattern Recognition by Artificial Touch," Pattern Recognition, Vol. 6 pp. 77-83, 1974.

[KINOSHITA-75] Kinoshita, Gen-ichiro, "A Pattern Classification by Dynamic Tactile Sense Info. Processing," Pattern Recognition, Vol. 7 pp. 243-251, 1975.

[NITZAN-80] Nitzan, David, "Assessment of Robotic Sensors," Workshop on the Research Needed to Advance the State of Knowledge in Robotics, April 15-17 1980.

[OKADA-77] Okada, T., and Tsuchiya, S., "Object Recognition by Grasping," Pattern Recognition, Vol. 9 pp. 111-119, 1977.

[PURBRICK-81] Purbrick, John A., "A Force Transducer Employing Conductive Silicone Rubber," Proc. 1st International Conference on Robot Vision and Sensory Controls, Stratford-upon-Avon, UK., IFS (Publications) Ltd., April 1-3, 1981.

[SALISBURY-81] Salisbury, Ken, Stanford Artificial Intelligence Laboratory, Personal Communication, May, 1981, and, Proc. of 1981 Joint Automatic Control Conference, Charlotsville, Virginia, June, 1981.