VISUAL FEEDBACK AND RELATED PROBLEMS IN COMPUTER CONTROLLED HAND EYE COORDINATION

Aharon Gill

Stanford University

Prepared for:

Advanced Research Projects Agency

October 1972

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-178

STAN-CS-72-312

VISUAL FEEDBACK AND RELATED PROBLEMS IN COMPUTER CONTROLLED HAND EYE COORDINATION

BY

AHARON GILL

SUPPORTED BY
ADVANCED RESEARCH PROJECTS AGENCY
ARPA ORDER NO. 457

OCTOBER 1972

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-178

COMPUTER SCIENCE DEPARTMENT
REPORT CS-312

OCTOBER 1972

VISUAL FEEDBACK AND RELATED PROBLEMS IN COMPUTER CONTROLLED HAND EYE COORDINATION.

by

Aharon Gill

A set of programs for precise manipulation of simple planar bounded objects, by means of visual feedback, was developed for use in the Stanford hand-eye system. The system includes a six degrees of freedom computer controlled manipulator (arm and hand) and a fully instrumented computer controlled television camera.

The image of the hand and manipulated objects is acquired by the computer through the camera. The stored image is analyzed using a corner and line finding program developed for this purpose. The analysis is simplified by using all the information available about the objects and the hand, and previously measured coordination errors. Simple touch and force sensing by the arm help the determination of three dimensional positions from one view.

The utility of the information used to simplify the scene analysis depends on the accuracy of the geometrical models of the camera and arm. A set of calibration updating techniques and programs was developed to maintain the accuracy of the camera model relative to the arm model.

The precision obtained is better than .1 inch. It is limited by the resolution of the imaging system and of the arm position measuring system.

This research was supported in part by the Advanced Research Projects Agency of the office of the Secretary of Defense under Contract No. SD-183.

The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Reproduced in the USA. Available from the National Technical Information Service, Springfield, Virginia 22151.

ACKNOWLEDGEMENTS

First I would like to thank my thesis supervisor, Professor Jerome Feldman, for his excellent guidance, especially his good sense of timing in being encouraging, critical or non-interfering at the right moments.

The accomplishment of this thesis was possible to a great extent through the devoted help of a large number of people at the Artificial Intelligence Project headed by Professor J. McCarthy whose imagination and vision created the project. The persons I would like to thank are too numerous, but I thank each of them individually.

This thesis is another step in the development of the Hand-Eye System. The system is a product of several years of team work, directed by Professor Feldman. During my work I have used ideas and programs developed by members of this team, especially: R. Paul, K. Pingle, V. Scheinman, I. Sobel and J.M. Tenenbaum. I enjoyed a very fruitful interaction with all the members of the group and would like to thank them all.

Thanks are also due to the other members of the reading committee, Professors M.E. Hellman and C. F. Quate for their useful comments.

I am very grateful to the Armament Development Authority of the Israeli Ministry of Defense which financed my studies at Stanford.

Last, but not least, I thank my wife, Rama, for her understanding, consideration and selfless devotion that made it all possible.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF NOTATIONS

In the following list E or Ei, where i is an integer, stands for an algebraic expression. The following "strange" symbols are used instead of subscripts, superscripts and other symbols which are usually not used in line.

SQRT(E) - the square root of E.
E1↑E2 - E1 raised to the power of E2.
ABS(E) - the absolute value of E.
E1*E2 - E1 multiplied by E2.
E1/E2 - E1 divided by E2.
INV([R]) - the inverse of the matrix [R].

The following abbreviations are used in the report:

c.c.s. - camera coordinate system
c.s. - coordinate system
c.t.c. - camera table coordinates
h.c.s. - hand coordinate system
h.t.c. - hand table coordinates
i.c.s. - image coordinate system
t.c.s. - table coordinate system
u.t.c. - user table coordinates

CHAPTER 1: INTRODUCTION

1.1 Background

The general field of research, of which this work is a part, is called Robotics. It rhymes with Aeronautics and Electronics and has the same connotations i.e. a field of engineering complex enough to deserve its own name. In my opinion it is a natural outgrowth of the field of Systems Engineering. Hence I would like to start with a discussion of systems in general and then describe the distinct features of robots.

We can informally define a system as follows: A collection of elements which has a goal. The following comments will give more substance to the definition:

(a) Every system, especially a complex one, is constructed of sub-systems, each with its own goal which is less inclusive than that of the parent system. If we subdivide the system into pieces which are small enough we will have devices, each having its own function. (The distinction between subsystem and device or between goal and function is intuitive and not formal and is done to help us in organizing our thoughts.

(b) We can divide each system into three major subsystems: I) A sensory subsystem which senses the relevant variables in the environment in which the system is operating, and the internal state of the system itself. II) A decision-making subsystem which combines the output of the sensory subsystem, the goal and stored information to decide what to do next. III) An effector subsystem which actually carries out the actions specified by the decision-making subsystem. One of the main aspects that make system design and analysis complicated and interesting is the fact that the environment presents to the system many random and time varying elements. Only some of them, hopefully the most important, are measured directly by the sensory subsystem.

(c) During the design phase and after, the analysis of system performance is done by using many tools: I) Mathematical analysis of the system model; II) Simulation of the model on a computer; III) Experiments with a breadboard model IV) Experiments with an operational system. (The research reported here, for example, consisted mainly of design and experiments of a breadboard model, the hand-eye system at Stanford, which is described later).

(d) When the structure of a system does not include any human elements (it is then called an automatic system), the goal that the system can accomplish is very specific and

is built very deep into the structure of the system. An airplane automatic pilot, for example, cannot be switched to pilot ships. The principle of operation, the method of design and analysis, etc, might be very similar, but the hardware and the software differ enough to make the switch impossible.

We now proceed with a definition of a robot: A robot is a versatile automatic system in the sense that it can achieve many different (probably related) goals at the same time or at different times, with no change or with very minor changes in its structure.

The following two examples will illustrate the meaning of the above definition. These were chosen since the development of these systems is being actively pursued by industry.

(a) Mars Explorer: Because of the time delay (minimum of 4 minutes round trip), direct manual control (as on the Russian lunar explorer) is not practical. The explorer should carry out by itself a variety of tasks given to it by its controller. For example: navigation, rock collection, soil sample analysis. The tasks are different but related in the sense that all involve interaction with the terrain: big rocks should be avoided, interesting rocks should be collected, rocks should be pushed aside to dig for soil samples, distinctive rock formations can help in navigation.

(b) Assembly line worker: In this case manual control will defeat the purpose of building one. The tasks include: mating parts, inserting and tightening screws, rivetting, welding, painting, [20].

The versatility of the robot demands versatility of the three major subsystems. Currently the most suitable candidate for a versatile decision-making subsystem is a general purpose digital computer. In the design of the other subsystems the temptation to build an anthropomorphic system is very strong. In addition to the often heard reasons concerning the optimization of mother nature over millions of years, I would like to add two more:

(a) Communication between robots and the goal setters would be much easier if they could share sensory experience.

(b) Accommodation: In the assembly line example the ability of the robots to move in spaces, to use tools, and to handle parts, all designed for human use, would be a major asset.

The major human sensory device is the eye. For this and other reasons (availability of T.V equipment for example) equipping the robot systems now in development with vision sense is universal practice. It is coupled with an arm and hand device (at Stanford [10], M.I.T, U. of Edinburgh [3], and in Japan: E.T.L [13] and Hitachi [6]), or with mobility (at Stanford [24] and S.R.I [16]).

Some of the problems encountered in the development of robots and which are dealt with in the research reported here will be described in the context of the robot project at Stanford.

1.2 The Stanford hand-eye system

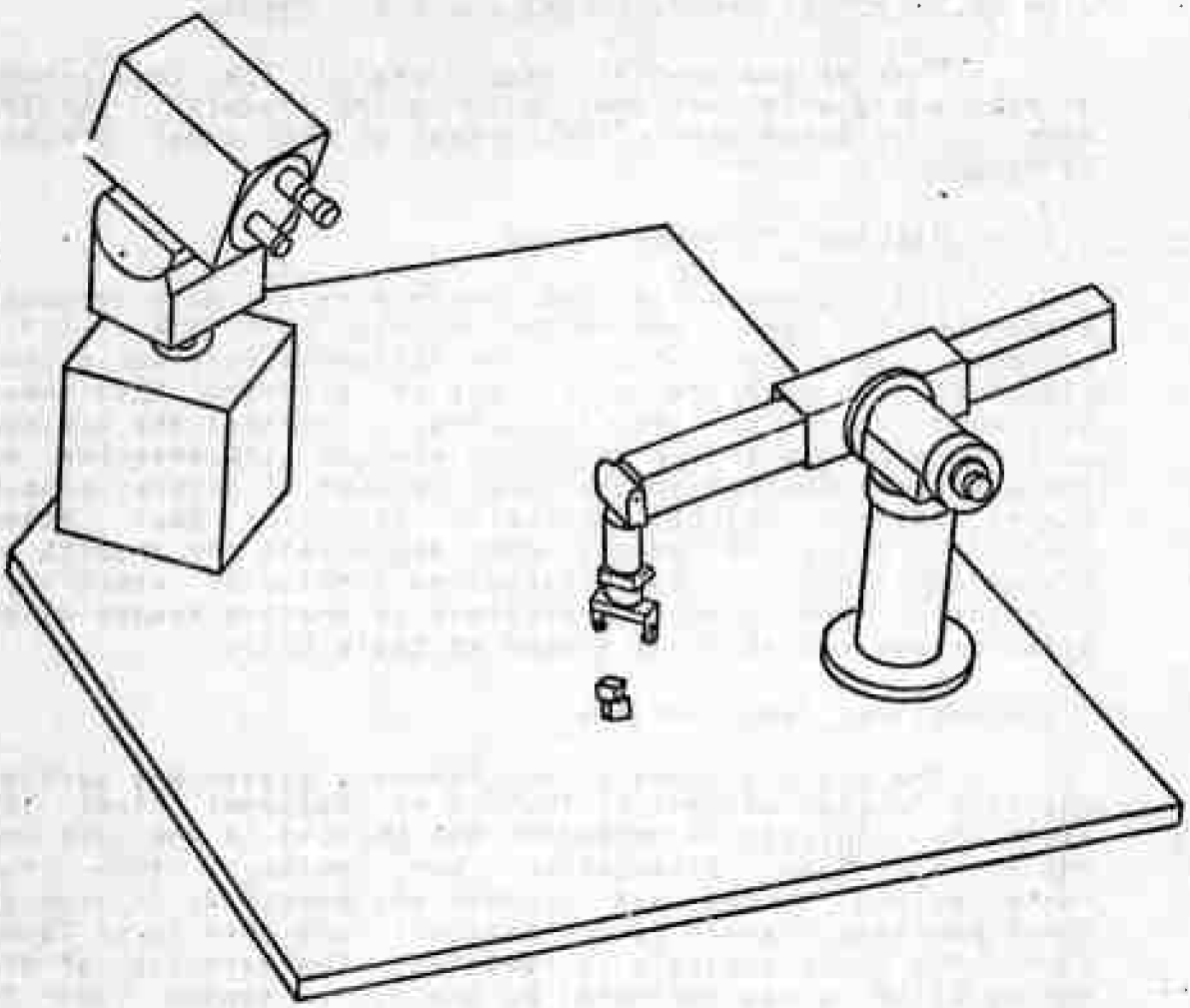
The hardware of the system consists of a computer controlled T.V camera and mechanical arm. Both are rigidly connected to a table which is the systems's work space (see Figure 1). Over the years a set of programs have been developed which do the following: control the various parameters of the camera, plan and execute trajectories of the arm [17], analyze scenes which consist of several simple planar bounded objects partially obscuring each other [19],[7], determine colors and manipulate the objects to change their position and orientation (including stacking). (Currently there is work in progress to analyze scenes which also include curved objects such as tools [1]).

1.3 Visual feedback problems

The various tasks of the hand-eye system are carried out in a "gross" sequential fashion as follows: First the scene is analyzed to determine the objects it includes and their position, orientation and color. Then the manipulations needed are planned and executed. If some of the parameters changed by the manipulations have to be found again, the scene analysis is repeated. The execution of the manipulation is not monitored by the vision sensor (the TV camera) in any way. This manner of operation is in principle relatively imprecise, uneconomical and sometimes "catastrophic". These somewhat strong allegations will be explained next.

Precision: We will explain the issue of precision by an example. The precision with which a stack of cubes is built is determined by the following parameters:

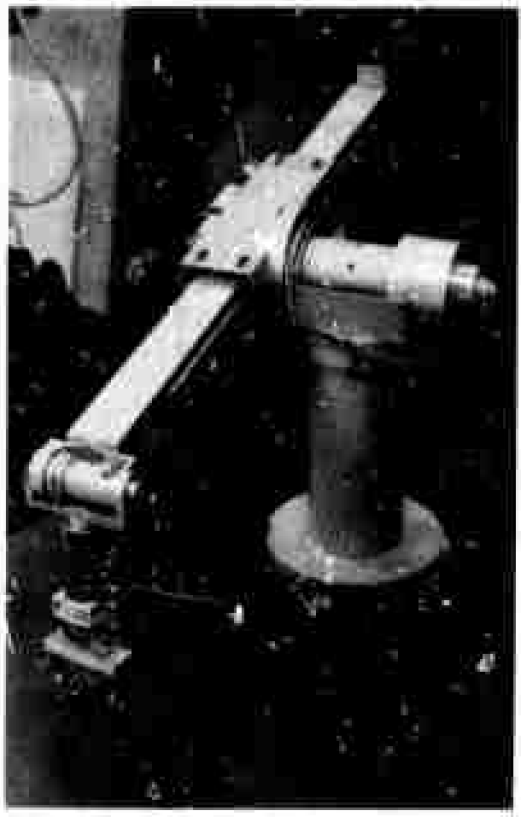
(a) The precision of locating the cubes initially. This location is computed from the location of their image in the image plane and the mathematical model of the camera that we have. There are five factors which affects this



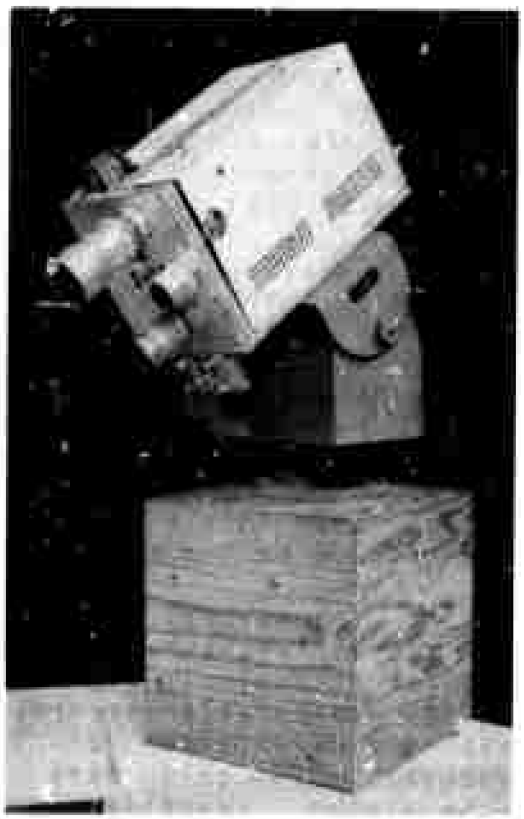
(a) GENERAL VIEW

Fig. 1.

HAND EYE SYSTEM HARDWARE
(Continued on next page)



(b) ARM



(c) EYE

Fig. 1. CONTINUED.

precision: I) The quality of the image. II) the precision of the scene analyzing routine which located the images of the cubes. III: The quality of the mathematical model of the camera, IV) The precision of the invariant parameters of the camera which are determined by a calibration program described later, V) The quality of measurements of the variable parameters of the camera such as the pan and tilt angles.

(b) The precision of manipulation. Here we have four factors: The first three are similar to III,IV,V above pertaining to the arm. The fourth is the precision of the arm servo program.

The resultant error may be more than we can tolerate.

Economics: When a scene is analyzed and then changed by manipulating some of the objects it includes, we know (and the program can know) fairly well how the new scene looks. A general scene analyzer, when called to analyze the new scene would not make use of this information. A scene analyzer that can make use of this information and look only for the remaining differences which are caused by the imprecision discussed above, would be much more economical.

Catastrophes: Consider again the cube stacking example. If the error is large enough so that the center of gravity of the top cube is outside the top face of the bottom cube, the top cube will fall off when it is released by the hand. The cube is released so that the hand can be moved away in order to simplify the scene for the general scene analyzer. Any scheme to avoid catastrophes will need the capability to analyze scenes which include the hand. Again the fairly good knowledge of the position and orientation of the hand could help in the scene analysis.

The natural solution to the above deficiencies is to use visual feedback. I call this solution "natural" because it is extensively employed by humans. The limiting factors of the manipulation errors, when visual feedback is used, are the resolution of the image and the arm servo. The accuracy in this case is inherently better than for "open-loop" operation. The problems generated by the "open-loop" mode of operation, and the posibility of using visual feedback as a solution have been known by researchers in robotics. However, the implementation of the solution had to wait for the accumulation of knowledge and technological know-how to reach the right stage. I was fortunate to start working with the hand-eye group of the Artificial Intelligence Project at Stanford when this stage was attained.

1.4 The goal an short description of the work.

The goal of the research reported here was to give the hand-eye system some visual feedback capabilities for the tasks carried out by the system at that time, i.e achieve precise grasping, placing and stacking of objects (mainly cubes).

The work was divided into three parts, developed in approximately chronological order, although changes were made as the work in later parts advanced:

(a) Development of a simple scene analyzer which can find lines and corners in a small part of the image frame utilizing as much information as is given to it at all levels of its operation. This analyzer, which we will call the corner-finder, is suitable for the visual feedback tasks.

(b) Development of calibration updating programs. These programs are needed to maintain the calibrated model of the camera at its best against deterioration over time. It is a general principle of feedback systems that the complexity and effort needed of the feedback part of the loop is inversely proportional to the precision that can be attained by the forward part of the loop. In our case this relation is not smooth since a small change of precision (especially when the precision is on the same order of magnitude as the dimensions of the objects handled) can mean a qualitative change in the nature of the feedback part of the loop. These programs make use of the corner-finder of part (a). The calibrated model, although partially deteriorated, is still good enough so that information supplied by it can be used by the corner-finder.

(c) development of visual feedback task programs using the corner-finder of part (a) and relying on the precision of the forward part of the loop maintained by the programs of part (b).

As an example we will give here a simplified description of the stages of operation when a cube (A) is grasped, with visual feedback, and then stacked on another cube (B) of the same size, again with visual feedback. (The numbers after each step refer to figure numbers):

(a) The hand is brought over A. (32(III)).
(b) The fingers are closed till one of them touches the cube as determined by touch sensors on the fingers. (32(IV).
(c) The hand is located in the image using the corner-finder and the error of its position relative to the

cube is computed.

(d) The fingers are opened, (32(V)), and the error corrected by incremental movement of the arm (32(VI)). Steps (b) to (d) are repeated till the errors are reduced below a given threshold. Then the cube is gripped. (32(VIII)).

(e) The top edges of the bottom cube are located in the image.

(f) A is brought above B. (35(IV)).

(g) A is placed on B using force feedback. (35(V)).

(h) The hand is located again in the image, and then the visible edges of the bottom face of A.

(i) The position errors are calculated, A is lifted, (35(VI)), and the errors are corrected (35(VII)). Steps (g) to (i) are repeated, till the errors are reduced below a given threshold. (35(VIII)).

Usually the feedback loops are executed only twice.

One of the main problems that we had to solve was the analysis of the complex scene, containing the hand and at least two more objects, to measure the small errors to be corrected. To achieve this goal the corner-finder was designed. Much effort was put into maintaining the accuracy of the calibration of the system, by designing the calibration updating programs and using other schemes described later.

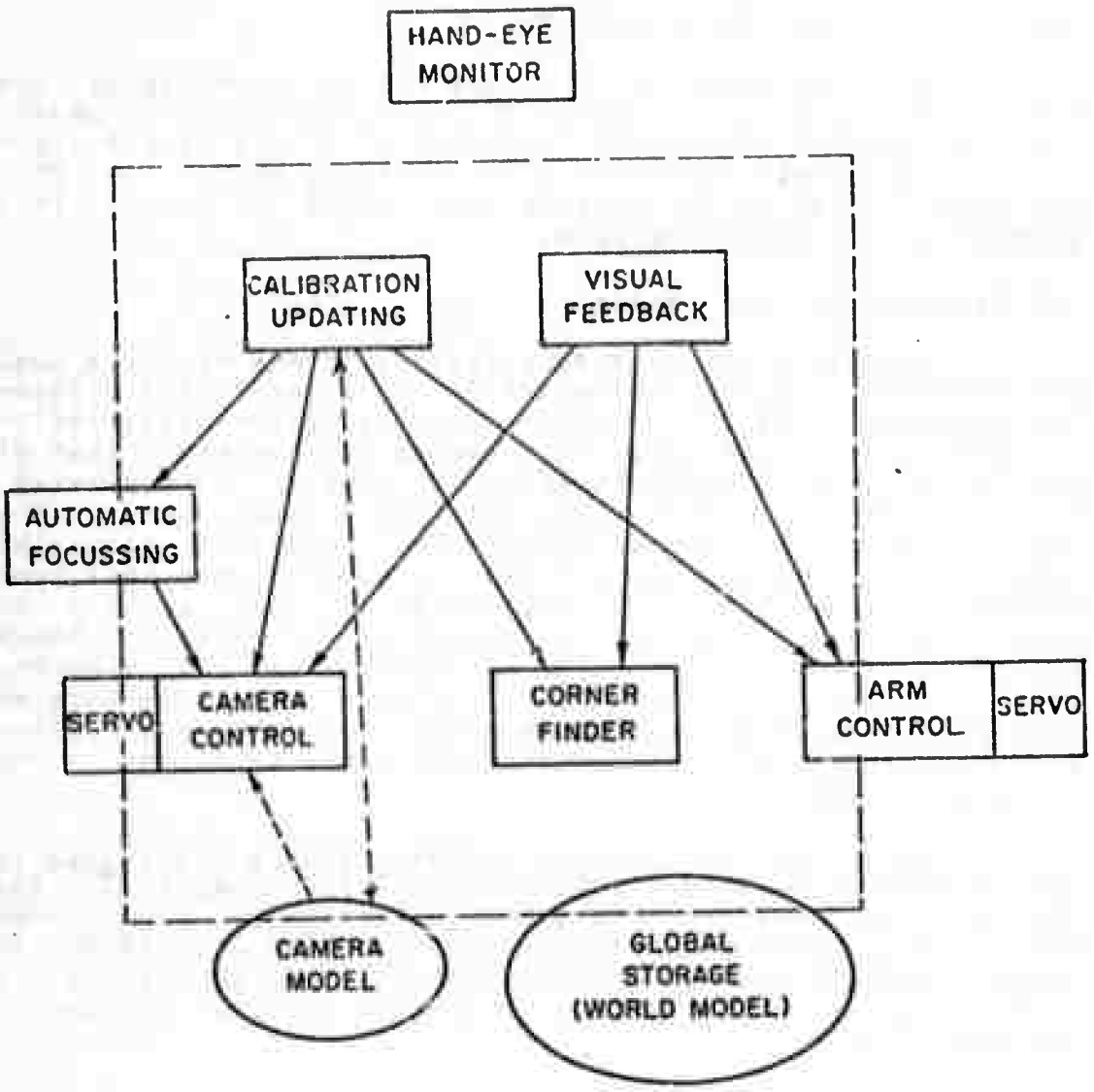
Two existing programs: the camera controller and the automatic focussing program were modified for use in parts (b) and (c) of the work. (See Figure 2).

1.5 Related work

The work described here is in part a direct outgrowth of the work done at Stanford by I.Sobel and ideas suggested by him [26]. Parts are extensions of the research done at Stanford by J.M.Tenenbaum [28]. The work is also closely related and makes use of programs developed by my co-workers at Stanford, especially R.Paul who developed the arm trajectory planning and control and K.Pingle who helped develop and maintain the utility programs and the hand-eye system monitor. (See Figure 2).

Two other efforts in visual feedback should be mentioned here:

(a) The heroic efforts of W.Wichman who five years ago at Stanford tried to develop visual feedback capability without the support and environment that we now enjoy. In Wichman's system a general scene analyzer was used to determine the error of stacking two cubes, after the hand was removed from the scene. Then the hand was brought in



—> Messages

--> Information store and retrieve

(All modules store and retrieve information from Global Storage)

Fig. 2.

ORGANIZATION OF PROGRAM MODULES AND STORAGE

again to correct the errors [29].

(b) Work done by the Artificial Intelligence and Robot group at E.T.L (Electrotechnical Laboratory) in Japan. The task carried out is the insertion of a long square block into a square hole [25]. The block is long enough so that the hand holding it is outside the field of view. This simplifies the scene analysis.

1.6 Structure of the report

Chapter 2 includes descriptions of the hardware used and software support. In chapter 3 the corner-finder program is described in detail. Chapter 4 contains the description of the calibration updating programs and chapter 5 that of the various visual feedback tasks. Chapter 6 summarizes the results and suggests possible future developments. The descriptions of the camera control and automatic focussing programs are left to appendices. Chapter 7 describes some general problems concerning the design of robot systems in particular and large scale programs in general. These problems, or examples of them, surfaced during the research. Many of them remain unsolved. In some cases, comments on the principles of solutions are given with the description of the particular implementation of these solutions in the current programs.

Guide to the casual reader: Readers interested in the corner-finder should read Section 2.1.2, Chapter 3, and Section 6.2. Readers interested in calibration should read Section 2.1, Chapter 4 and Section 6.3. Those interested in visual feedback and not in the details of how the scenes are analyzed, should read Chapter 2, Section 4.1.2, Chapter 5 and Section 6.4.

CHAPTER 2: HAND-EYE SYSTEM HARDWARE AND SOFTWARE SUPPORT

2.1 Hardware and Interface

2.1.1 General notes

We have two goals in the following hardware description. One is to give a general idea of the hardware involved to facilitate the understanding of the work described. The other is to describe in detail the subsystems and parameters which have had a marked effect on this work.

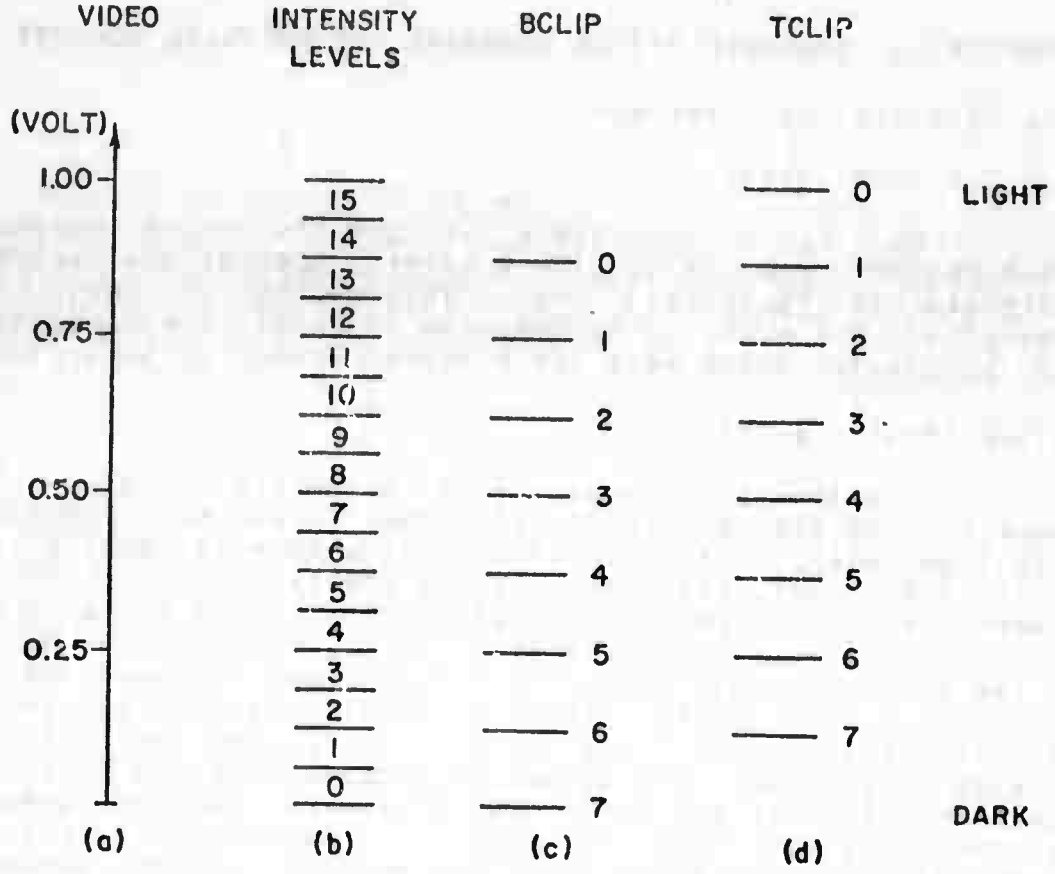
2.1.2 Imaging system.

The imaging device is a commercial vidicon T.V camera with standard raster scan (525 lines, 60 fields/sec, 1:2 interlace). The video signal has a dynamic range of 0-1 volt (amplifier saturation and a special limiting circuit keeps the signal from being higher). The p-to-p noise in the video signal is approximately 60 mv, hence the signal to noise ratio (maximum signal to r.m.s noise) is approximately 60:1. The video signal is sampled at the rate of 333 samples/line and 256 lines/field. Only one of the two fields is sampled. The number of samples/line was chosen so that the vertical and horizontal distances between samples on the vidicon face is equal. Sometimes we call the sampling points, "cells" since the sampled values actually correspond to the average intensity absorbed by a small cell on the vidicon face.

The samples are converted to 4 bit numbers (0-15), packed 9 in a word and shipped directly to core memory via a fast (24 Mbits/sec) channel. The channel bit rate limits the number of bits/sample since no local fast buffer is used. The sampling resolution is approximately 600/inch or the vidicon face which corresponds to 1.5 mrads with a 1 inch focal length lens.

The digitizing process is flexible in the sense that we can specify to the A/D converter which interval of the full video signal dynamic range (1v) to use as its own full range. The interval is specified in multiples of 1/8 volts. The location and width of the interval is specified by two parameters TCLIP and BCLIP which will be called the "clips", (See Figure 3). By using 8 contiguous intervals each 1/8 volts wide, we can effectively have 6 bits. This is close to the maximum useful number of bits because of the signal-to-noise ratio mentioned above.

We should note that the vidicon camera was designed with tolerances suitable to an adaptable human looking at a monitor connected to the camera in closed circuit. The



(a) Video signal

(b) Corresponding intensity levels with clips open (TCLIPS = 0, BCLIP = 7).

(c) BCLIP settings

(d) TCLIP settings

Fig. 3

CLIPS SETTING

tolerances that we would like it to have as a measuring device are much stiffer.

In order to conserve storage space we can specify which part of the total number of 256*333=85248 samples to store. This part is called a "window" and is specified by the first and last line (FLINE and LLINE) and the first and last sample on a line (LSIDE and RSIDE). The window is always rectangular and parallel to the scanned reactangle. The number of lines the window includes is called its "height" and the number of samples/line is called its "width". Each sample point can be specified by its coordinates in the "image coordinate system" (i.c.s.). (See Figure 4). The maximal size window, i.e the one which includes all the samples, is called the "frame". The 4 bit number stored for each point is called simply the "intensity" at this point.

From time to time we mention also "actual intensity" which is the intensity of light reflected from the object in the direction of the camera. The actual intensity is determined by the lighting conditions and the reflective properties of the object and, to a lesser degree, the reflective properties of the surrounding background and other objects. This intensity could be measured by a light-meter situated near the camera.

For a given lens and filter (we have a number of each) there are two factors which most affect the video signal level. One is the aperture which is not controlled by the computer and is usually left full open to get the maximum amount of light. The other factor is the vidicon target voltage. When the automatic target voltage setting circuit (also called the automatic sensitivity circuit) is operated it changes the target voltage so that the video signal corresponding to the highlights of the scene will be just below the maximum level (1 volt). When more flexibility is needed, the target voltage can be changed by the computer [2$_8$]. In this case we have to take care not to "burn" parts of the vidicon face.

2.1.3 Camera control

The following camera parameters can be changed by the computer. They all have read-out connected to the computer so that the change is controlled.

(a) The lens used can be changed by rotating a turret with 4 lenses of different focal lengths.

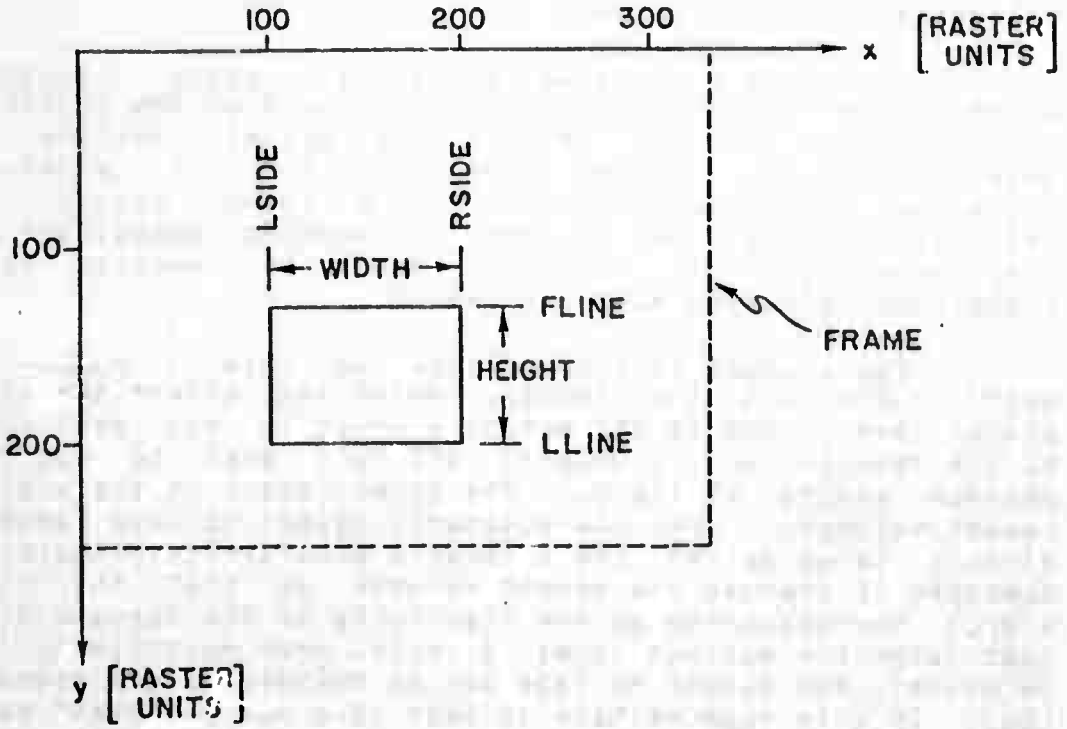


Fig. 4.

IMAGE COORDINATE SYSTEM (I.C.S.) AND WINDOW SPECIFICATION

(b) The distance between the lens center and the face of the vidicon can be changed. This is done by moving the whole vidicon. This distance determines the range at which the camera is focussed. This method of focussing was chosen so that one mechanism is needed instead of the four that would be needed to control the focussing ring of each lens separately. Also, the longer focal length lens can be focussed at shorter ranges than is normally attained.

(c) The pan and tilt angles of the camera can be changed. This produces a translational as well as rotational movement. (See Figure 5). For example: when using the 1 inch lens, when the camera is panned so that an object which appeared at the extreme left of the frame will now appear at the extreme right, the lens center is translated by approximately 2.5 inches.

The pan, tilt and focus servoes are of the sampling, on/off with threshold (dead-zone) type, with constant rates (.06 rad/sec in tilt, .13 rad/sec in pan and .012 inch/sec in vidicon movement). The sampling rate is 60 samples/sec. The thresholds (full) correspond to 1/5 depth-of-field for focus and 3 mrads in pan and tilt.

(d) A color filter, which is inserted between the lens and the vidicon face, can be changed by rotating a wheel with 4 filters (red, green, blue and clear).

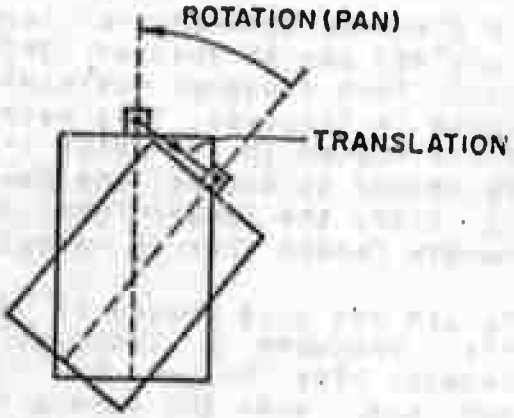
2.1.4 Arm hardware and control

The arm has 6 degrees of freedom, which means that within structural constraints it can position the hand at any location in its working space and with any orientation [23].

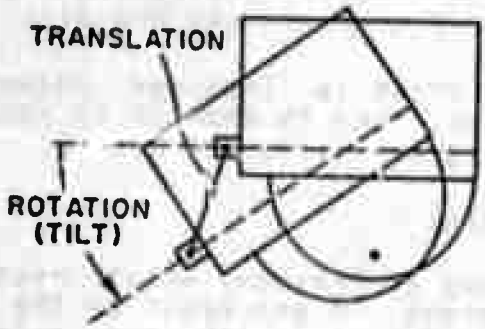
The control of the arm is much more sophisticated than that of the camera [17],[18], and will be described here very briefly.

The arm has two rotary joints ("shoulder" joints), followed by a prismatic joint, followed by three more rotary joints ("wrist" joints), ending in a hand. All joints are actuated by D.C electric motors. The torque generated by each motor is controlled by varying the voltage pulse width supplied to it. On each joint there is an "infinite" resolution potentiometer supplying position information to the computer via an A/D converter. On each joint there is also a brake to hold it in its position after it stops.

The potentiometer readings are sampled 60 times/sec. The position and computed velocity are compared to the desired values for this instant of time, and then the



(a)



(b)

(a) Top view

(b) Side view

Fig. 5.

PAN AND TILT ROTATIONAL AND TRANSLATIONAL MOVEMENTS.

necessary torque is computed for each joint taking into account the varying weight and moment of inertia loading each joint. At the end of the trajectory the remaining error is nulled.

Typical trajectory execution takes less than 4 seconds. The accuracy of positioning over the table is .1 to .2 inch. The repeatability of the servo,(limited by the resolution of the potentiometers and the A/D converter), is .03 to .05 inch.

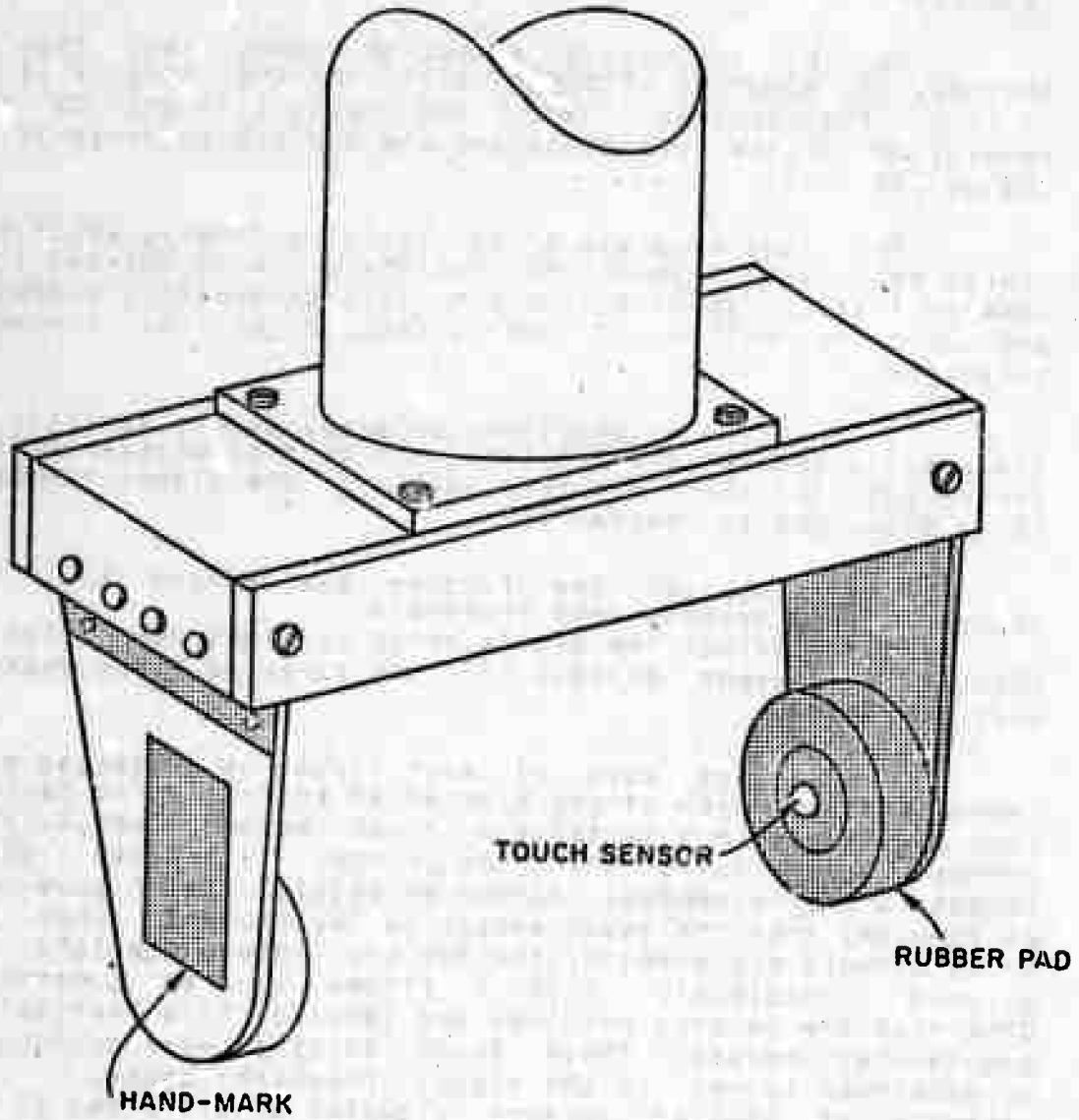
The hand consists of two opposing fingers which are opened and closed together so that the midpoint between them remains fixed relative to the arm. This generates a sweeping and gripping motion in the direction of the fingers' movement.

In addition to position, orientation and control of the opening between the fingers, the control program offers other options which are very useful in the visual feedback tasks described in Chapter 5:

(a) Grasping: The fingers are closed till the opposing force exceeds some threshold.
(b) Placing: The arm is moved straight down (with or without a grasped object) till the opposing force exceeds some threshold.

The inside part of each finger is protected by a rubber cushion which serves also as an anti-slipping device. Each of the cushions contains a touch sensor, actually a contact which opens when the finger is pressed lightly (about .1 oz. is needed) against an object. (See Figure 6). We then say that the touch sensor is "activated". When the touch sensors are enabled, the arm and fingers motions are stopped immediately when a finger touches something. Otherwise the sensors readings are ignored. This "reflex" is the touched object. These touch sensors were devised by V. Scheinman to help in the visual feedback tasks. An ensemble of similar sensors situated on all sides of the fingers and at selected points on the arm could be used as a self-protecting and exploring device [2].

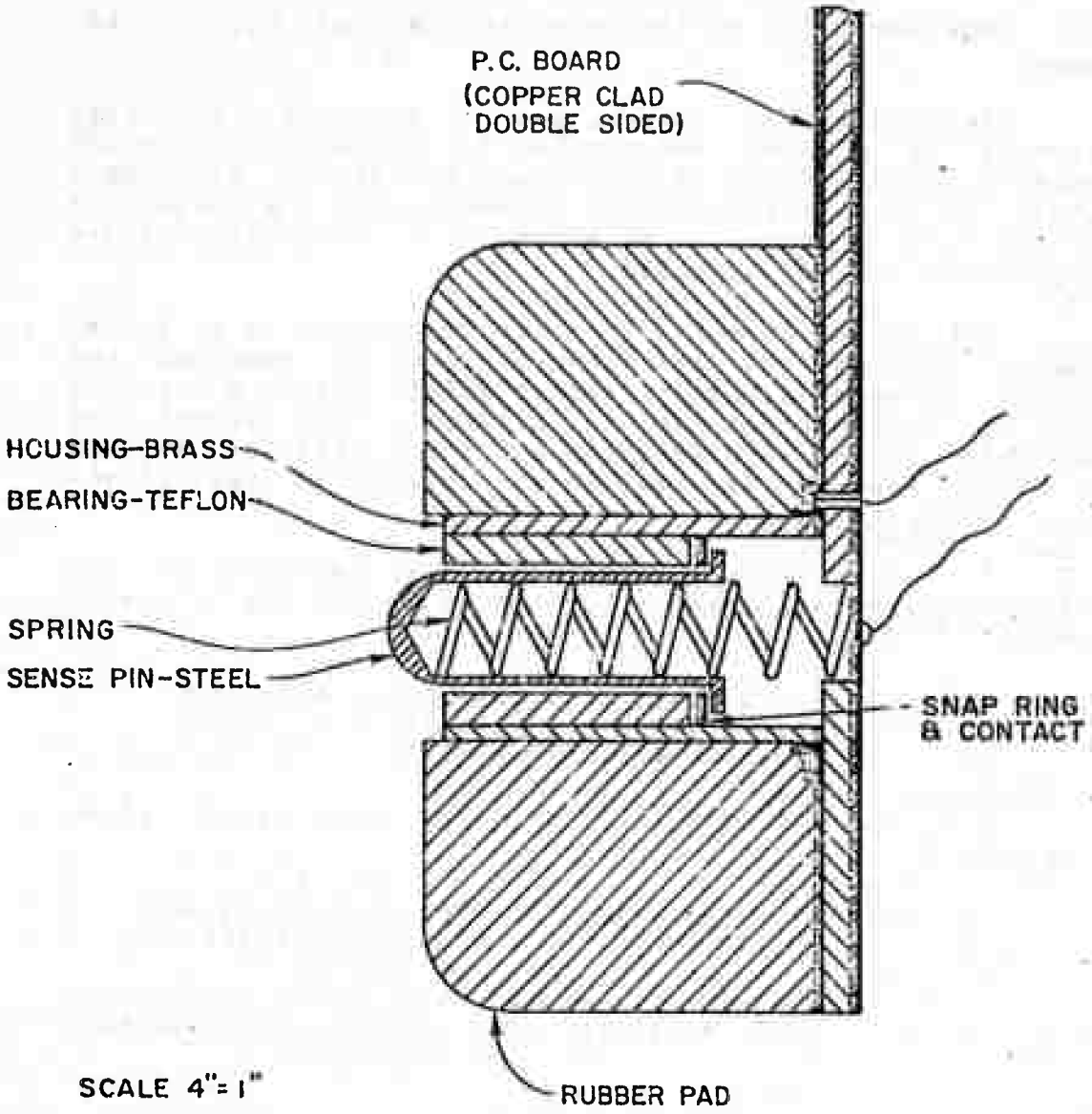
A black rectangle is painted on the outside face of each finger. We will call it the "hand-mark". It helps in the recognition of the hand in the scene.



(a) GENERAL VIEW

Fig. 6.

TOUCH SENSORS
(Continued on next page)



(b) SECTION VIEW

Fig. 6.
TOUCH SENSORS
(Continued)

The operation of the arm is carried out in two stages:

(a) Trajectory generation: A sequence of needed movements is planned and checked for violation of its own structural constraints, going through the table, etc. Each segment of the trajectory is planned under the assumption that the last segment will be successfuly completed before this segment will be executed.

One of the segments that can be planned is a "wait" segment in which the arm control program receives and executes commands directly without going through a planning phase. The motions that can be realized in this segment are those which do not significantly change the position of the arm. This feature is used in the correction phase of the visual feedback tasks.

(b) Trajectory execution: The segments of the trajectory are executed in turn until one of the following happens:

(1) The last segment is successfully executed.
(2) The program reaches a planned wait segment.
(3) The program detects an error or malfunction of the hardware.
(4) One of the touch sensors is activated (after being enabled).

In the last two cases the control program enters a wait state which is equivalent to a planned wait segment.

When a "proceed" command is given in the wait state, the execution of the trajectory is resumed in the next segment.

2.2 Software environment and support

2.2.1 General notes

In principle the software environment in which the set of ideas and programs evolved should be irrelevant, since the ideas and solutions are hopefully general enough to have scientific merit of their own. In practice however it is not so and the ideas and solutions are at least influenced by the software environment. We can look on the software environment as a part of the general environment within which the hand-eye system has to operate, which also includes such factors as lighting conditions, electromagnetic interference, mechanical vibrations of the structures, etc.

Because of its relevance a short description of the software environment and support is given in the following sections.

2.2.2 The time-sharing system

The Artificial Intelligence Lab computing facility includes the following components:

(a) Main processor: Digital Equipment Corp.'s PDP-10/32.

(b) Auxiliary processor: DEC's pDp-6, which handles the I/O devices. The hand and camera serv programs are executed on this processor in real time and i parallel to the main program executed on the main processor. An interrupt clock is connected to the auxiliary processor and makes it possible to execute the programs which are in real time mode at the rate of 60 times/sec.

(c) Fast core memory of 256 Kwords (36 bit words). The memory can be accessed directly by both processors, swapping channel and various other channels. The size of the user programs is currently limited by the system to 90K.

(d) The system is operated in a time-sharing mode only. All the jobs which are active but do not have space in the fast core are swapped to a special disk (at a rate of 600 word/msec). The number of jobs sharing the system are rarely less than 10 and frequently more than 30.

(e) The back-up storage is a set of disks where files are kept for each user. These files can be programs, their compiled versions, dumps (i.e core images), results etc. Transfer between the discs and core memory is 2.5 times slower than swapping.

(f) Hard-copy output devices: A line printer and a digital x-y plotter.

(g) The main user I/O device is a keyboard and a display monitor. There are two kinds of display monitors: one is a random access CRT type and the other is a raster scan TV monitor type. Both can display alphanumeric text and line drawings. The line drawings are more pleasing on the CRT type and vice-versa for text. Currently there are six CRT type displays (Information International Inc.) and sixty-four TV type displays (Data Disc Inc,). In addition a number of teletypes of various kinds are connected to the system.

2.2.3 Languages

The high level language used in the programs described here is SAIL which is a variant of ALGOL with an associative data structure and its manipulations added [27].

When speed is important (although the compiler is efficient), or when writing parts of programs to be executed on the PDP-6, assembly language is used.

A number of LISP variants, FORTRAN IV and other languages are available in the system but were not used in the current work.

2.2.4 The Hand-Eye Monitor [9]

Most of the programs needed to run hand-eye tasks, for example the pan/tilt calibration updating program described in Chapter 4, are too large to fit into the fast memory. The above mentioned program's size is 130 Kwords. More complicated programs need close to 500 Kwords. In order to be able to run, the programs are broken into parts, called "modules", that are small enough to fit into the memory. Most of these modules are general enough to be used in various tasks.

These modules are run as pseudo-jobs in the time sharing system. Usually each job has its own I/O device. In order not to consume a large number of displays (up to 8 currently) and for the convenience of the user all the modules can communicate with the user via a single I/O device. Usually it is a CRT type display monitor since line drawings are heavily used as condensed and meaningful outputs.

The communication between the modules is done in two ways:

(a) Sending "messages" between the modules. Each message includes the sender and intended recipient names, the name of the procedure the sender asks the recipient to execute and the parameters needed by this procedure.

(b) Altering the values stored in memory cells in a special area of the memory, called the "world model", which can be accessed by all modules. The variables in this part of the memory are called "global" variables.

The program that handles all the communications among user, modules, the time sharing system, I/O device and the world model is called the Hand-Eye Monitor. To carry out a specific task the following modules are usually used: A "driver" module which "knows" about the task, a number of utility modules (e.g. the corner-finder described in Chapter 3, camera controller, color-finder, hand controller) and the hand-eye monitor. The construction of a general purpose driver which would be able to plan and execute a family of tasks is a major concern and activity of A.I. research.

CHAPTER 3: CORNER-FINDER MODULE

3.1 Introduction

3.1.1 General notes

The purpose of the corner-finder is to find lines and corners (which are the main features of planar bounded objects) in a small area of the frame, utilizing information about the features to the extent given to it. As such, it is not a general scene analyzer (even in the context of planar bounded objects),and although it can be used as part of one, it will be uneconomical to do so. The corner-finder operates by analyzing part of the area (a window) at a time and moving the analyzed window in a controlled search pattern when needed.

Most of the scene analyzers used in robot projects operate in two steps: (a) Extract a line drawing from the image, (b) Analyze the line drawing. The lines are actually boundaries between image regions of different characteristics. Two main types of scene analyzers using simple intensity information have been developed over the years:

(a) The "gradient follower" type looks for boundaries of regions by analyzing intensity gradients at image points. Boundary points are points with relatively large gradients. Then a direction perpendicular to the gradient is followed to get the next point to be analyzed. The advantage of this scene analyzer is that it does not have to process all the points in the frame. It fails at points where the gradients are weak, at very sharp corners or when two lines come close together [12],[19].

(b) The "region grower" type aggregates points based on some similarity criterion to form regions. Rules are then used to merge regions into bigger regions. Boundary points are defined simply as points adjacent to two regions. The disadvantage of this type is that every point in the frame is processed several times [4].

The corner-finder uses ideas from both these types. It makes rough checks on the existence of regions in the analyzed area. For this purpose each point within the area is processed simply to form the intensity histogram of the area. Then it follows boundaries of regions by using a dissimilarity criterion. No gradient type processing is used so that continuity is not lost at points of weak gradient, sharp corner, etc. General scene analyzers do not use any prior information because there is no reason for them to assume the existence of such information. On the other hand

the corner-finder described here uses prior information down to its lowest levels. It can use as much information as given to it but does not require complete information for proper functioning. The design philosophy is to use and check against prior information at the earliest possible moment.

These introductory remarks are followed by the definition of terms used throughout and a more detailed description of the underlying assumptions and design philosophy. Then we present the details of the workings of the analyzer (FINDIMAG) and the search controller (SRCHIMAG). Last we describe the parameters controlled by the user, the format of the message procedure and program output.

3.1.2. Definitions of some terms

The term "corner" is used to describe a number of related entities :

(a) A corner of a physical planar bounded object, including the vertex, the edges and the faces associated with this corner. The "location" of the corner refers to the location of the vertex given by its three coordinates in a coordinate system tied to the table. (See Figure 7(a)).

(b) The image of the physical corner as encoded in the stored intensities. (See Figure 7(b)).

(c) An approximate line drawing of the image which is the output of the corner-finder program. The line segments emanating from the vertex and truncated by the window perimeter are called again "edges" of the corner. The "location" of the corner refers to the location of the vertex of the line drawing given by its two coordinates in the image coordinate system. We distinguish between two kinds of corners : "simple" corners which have only two edges, which will be called "sides", and "complex" corners which have more than two edges. (See Figure 7(c) and 7(d)). The corner-finder can find only simple corners directly. Complex corners can then be constructed from simpler corners. Generally, the vertices and edges of simple corners found in the image will not completely coincide even if the simple corners are parts of the same complex corner. Therefore we will merge them to form a complex corner if they are "close" (within some tolerance), and especially if there is some external information which indicates the existence of a complex corner rather than that of several separate simple corners. (See Figure 7(e)).

From here on the word corner by itself will refer to a simple corner of the line drawing.

The "inside" of a corner is the region of the image bounded by the two sides of a corner and including the angle between the sides which subtends less than π rads. In an image of a planar bounded convex body (not obscuring or obscured by other bodies) the inside of any corner is the image of a part of the body and the outside is the image of the surrounding background.

The side which is in a counter-clockwise direction from any point inside the corner is called the "first" side and the other is called the "second" side. (See Figure 7(d)).

When the image includes edges only and no corners the line drawing will consist of separate lines only. We call a line or corner in the line drawing a "feature".

We define the "form" of a feature as follows: If it is a line, then the form is the orientation of the line in the image. If it is a corner, then the form is the orientation of both sides as given for example by two pairs of direction cosines.

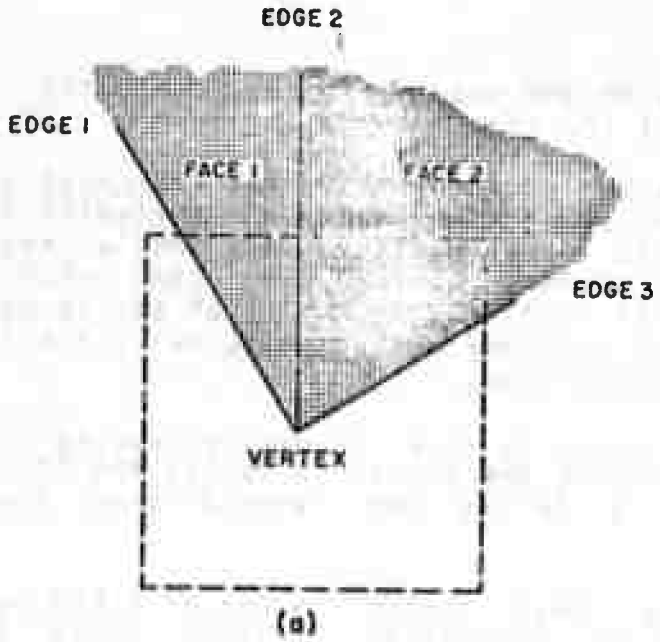
3.1.3 Basic assumptions and design philosophy

The following assumptions guided the development of the corner-finder. They are not all necessary conditions for its operation or success.

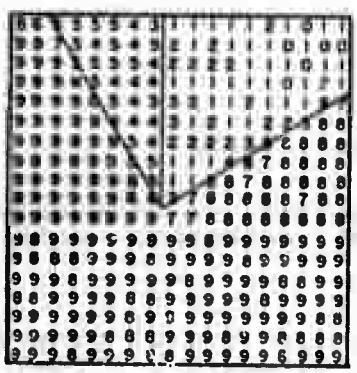
The most important assumption is that some of the properties of the corner (e.g. location, form and orientation, relative inside to outside intensity) are known at least approximately. They are known either because the properties of the object which this corner belongs to are known (for example: the hand or a specific cube), or because this corner was found before by the same or similar programs. The reasons that these properties are known sometimes only approximately are of the following flavours:

(a) Incomplete coordination between the hand and the eye, as for example when looking at the hand or at an object which was moved by the hand. This problem and some solutions are disoussed in Chapters 4 and 5.

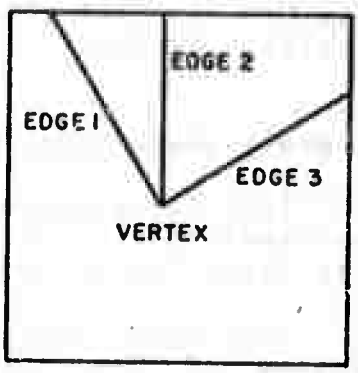
(b) The pan/tilt head of the camera moved, and we cannot predict accurately the new location of the corner in the image because its range from the camera is not known and the camera head movement causes translation in addition to



(a)



(b)



(c)

(a) Corner of an object

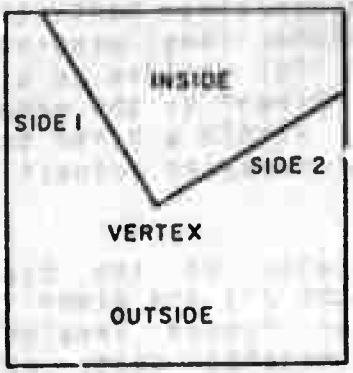
(b) Intensity map of the image of the corner

(c) Line drawing of the corner (complex corner)

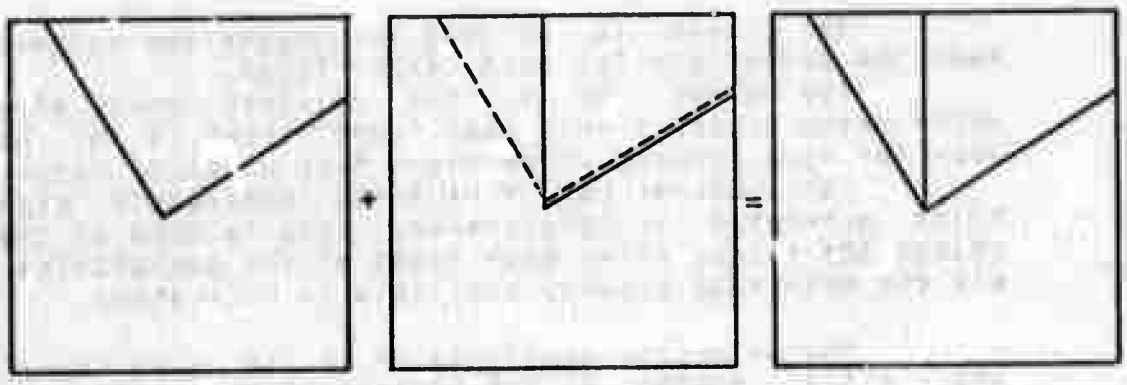
Fig. 7.

ILLUSTRATION OF THE DIFFERENT MEANINGS OF THE TERM: CORNER

(Continued on next page)



(d)



(e)

(d) Simple Corner

(e) Construction of a complex corner from two simple corners.

Fig. 7.

ILLUSTRATION OF THE DIFFERENT MEANINGS OF THE TERM: CORNER
(Continued)

rotation because the lens center does not lie on either the pan or tilt axes.

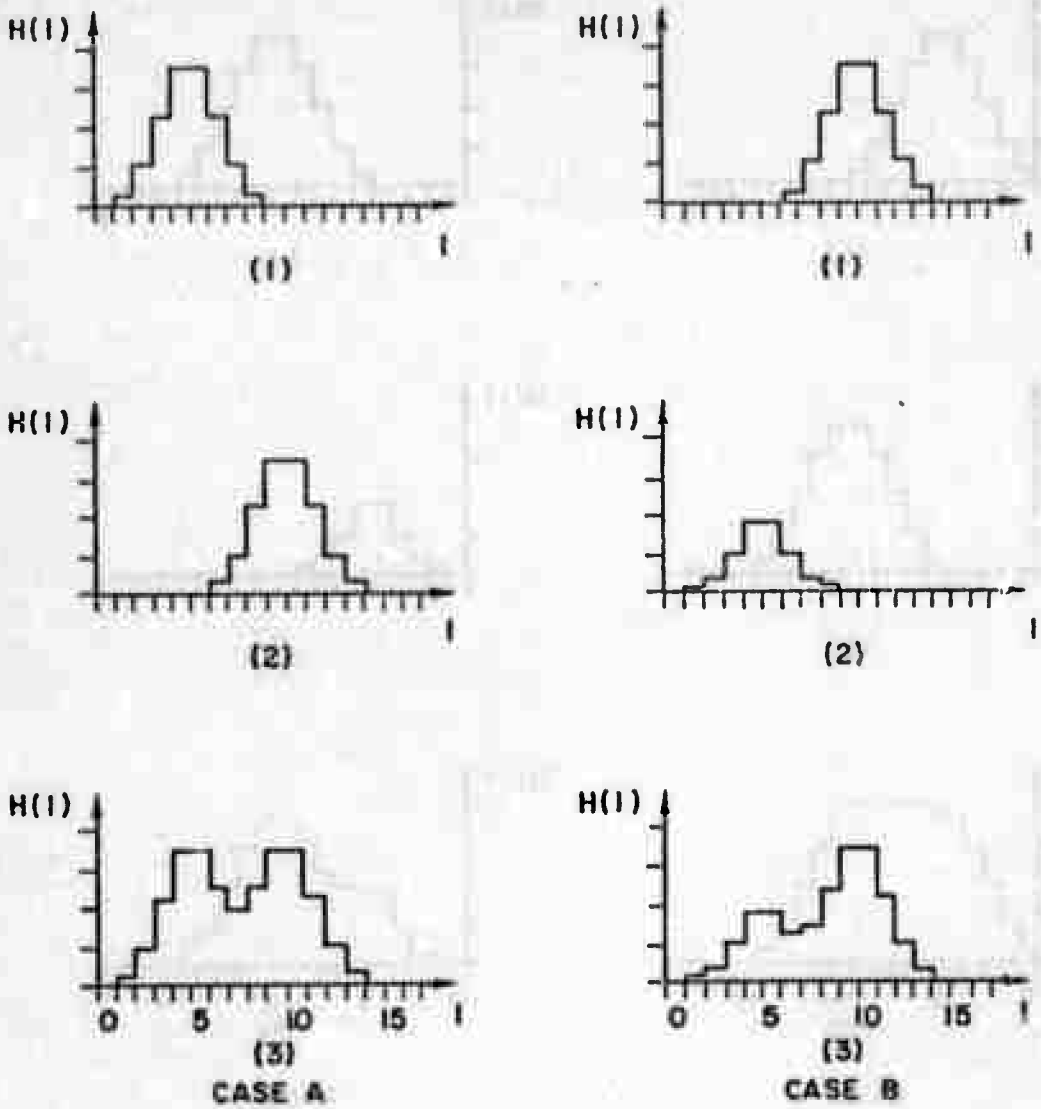
Not all the properties have to be given to the program. The user or a higher level program can give as many of the properties as he (it) decides to give. Actually the properties are not only "given" to the program, but the user can "demand" a match, within a given tolerance, of these properties and the actual measured properties of the corner found.

The current version of the program rejects any corner which does not match all the properties demanded to be matched. Another or future version might give as an answer the corner which matches best, according to some given criterion. This version would take more processing time and would become rather complex when the corner is searched for by moving a window in the frame. Using the first version instead of the second saves processing time (at the cost of flexibility and generality) at least in the three following ways:

(a) There is no need to compute the criterion and make the comparison for each corner found.
(b) After one or the required number of corners which match properly have been found, there is no need to look for more corners which might have a better match.
(c) A corner can be rejected immediately after the first mis-match is discovered. This is done by trying to reject the corner after each stage of the computation using all the knowledge already available at this stage.

The principle mentioned in (c) is also applied in the initial stages of the processing where the program is trying to reject the hypothesis that there is any corner at all in the analyzed window. For a trivial example, when it is found that the intensity is almost uniform over the window, the hypothsis is rejected.

A second assumption, which is a necessary condition, is that each region, defined as a contiguous part of the analyzed window bounded either by the perimeter or by edges, will have almost uniform intensity which is distinct from those of neighbouring regions in the following sense: In the intensity histogram there is at least one noticeable (see description of the block SLICE in Section 3.2.2) minimum between the intensity levels. (See Figure 8). The intensity is not completely uniform for the following reasons: slight differences in the reflecting properties of the surface and lighting, noise in the camera and quantization of an intermediate intensity level into two adjacent quantization levels by the A/D converter. This



Cases A and B - Histograms with noticeable minima

(1) Hypothetical intensity of region 1.

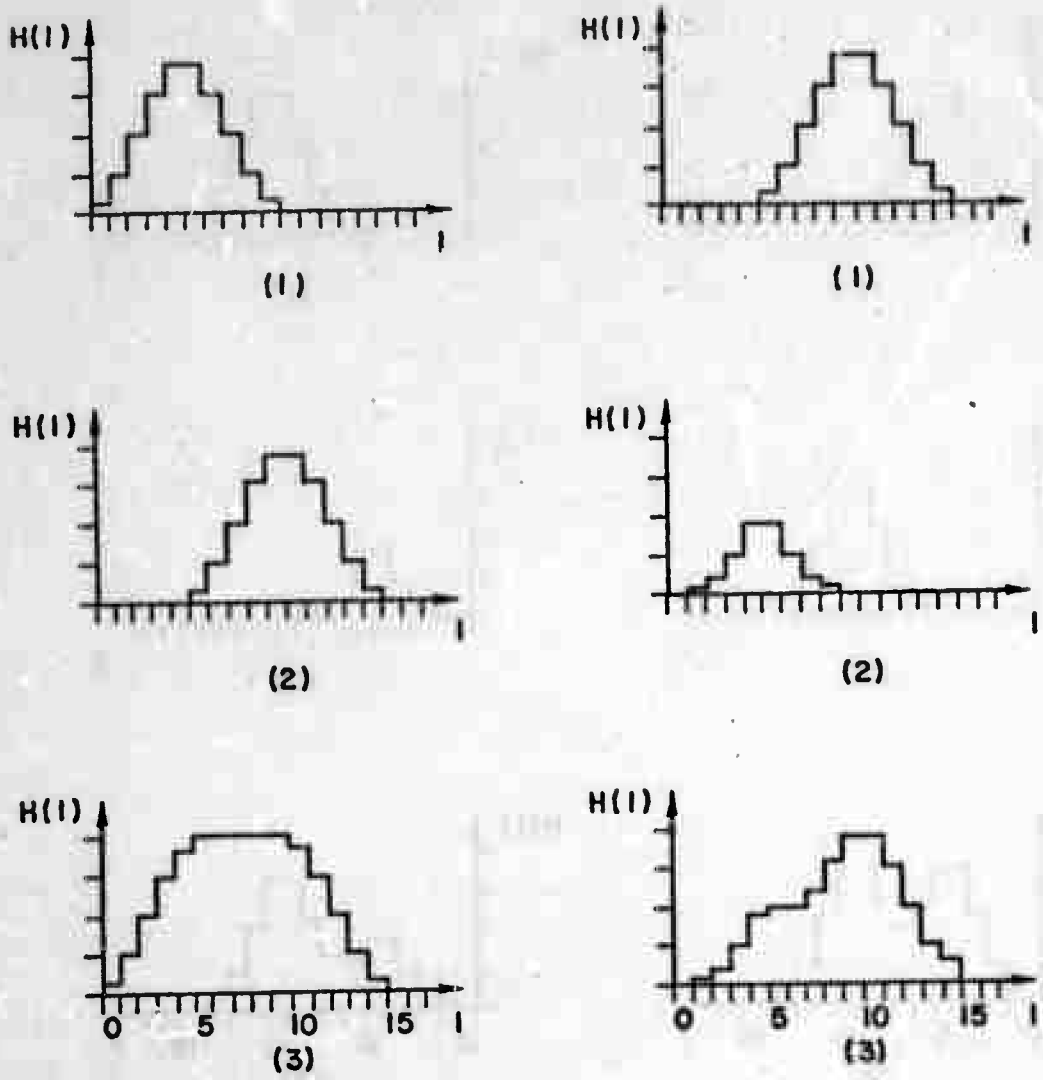
(2) Hypothetical intensity histogram of region 2.

(3) Combined intensity histogram of the window.

Fig. 8.

EXAMPLES OF HISTOGRAMS

(Continued on next page)



Cases C and D - Histograms with non-noticeable minima

(1) Hypothetical intensity histogram of region 1.

(2) Hypothetical intensity histogram of regions 2.

(3) Combined intensity histogram of the window.

Fig. 8.

EXAMPLES OF HISTOGRAMS
(Continued)

assumption influences the choice of window size. (See Figure 9). In some cases the presence of a simple corner, which is a part of a complex corner, can be inferred from other simple corners found in the window. See Figure 10 for an example. This assumption also influences the choice of window size. In most scene analyzers that look for polygons in the image, straight line segments are first fitted to the set of boundary points. From these line segments the presence and locations of corners (simple and complex) are computed. Because of the above assumption, the corner-finder will try to fit either one line segment or two connected line segments (i.e. a simple corner) directly to a set of contiguous boundary points found.

A fourth assumption is that the image of the corner occupies more than a certain fraction of the window area. When a small region is found it is rejected because it is assumed that either it is a noisy patch or that it is an honest-to-God corner but that we do not see enough of it and that it will be subsequently found again when the window is moved around in the frame in search of the missing corner.

We distinguish between two kinds of scenes: One is relatively simple, for example when looking at the hand alone, (see Figure 28). The other is cluttered, for example when looking at two stacked cubes, the top one still grasped by the hand, (see Figure 35(V)). When we are looking for more than one feature in the window, we implicitly assume that the scene is cluttered. In this case we do not like to search with the simple minded search strategies described later in Section 3.2.3 and leave it to the user or higher level programs to tailor their own search strategies to the cluttered scenes they expect.

Some comments about window size: The window size which is regularly used has a dimension of 18*18 raster units. When the 2 inch focal length lens is used it corresponds to a field of view of approximately 1 degree which incidentally is the field of view of the sensitive part of the human eye, the fovea. The fovea however has about 5 times more sensing elements in the same field of view [5]. We should note however the human ability to resolve between pairs of lines that are closer than the distance between the sensing elements. Carrying the above analogy a little farther we can say that moving the window in the frame is similar to the movement of the eye in the head, while moving the camera is similar to rotating the head.

This size of the window was chosen in order to fulfill the assumptions that the window is smaller than the image of the object so that each line or corner intersects

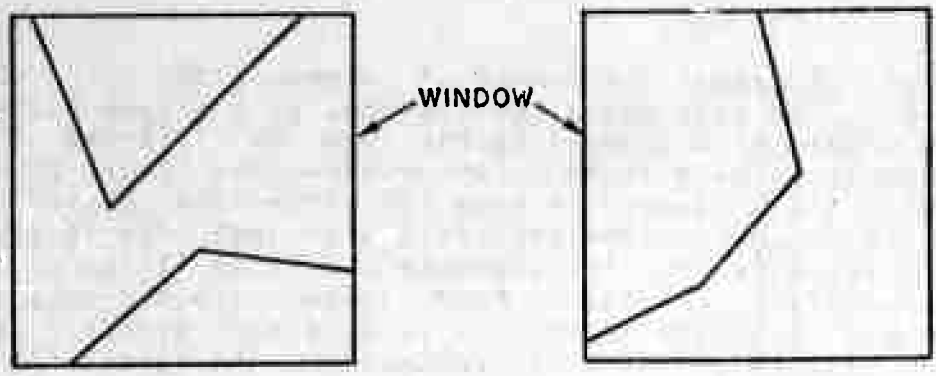


Fig. 9.

EXAMPLES OF SIMPLE CORNERS WITH SIDES INTERSECTING THE PERIMETER (CASE A), AND NON-INTERSECTING (CASE B)

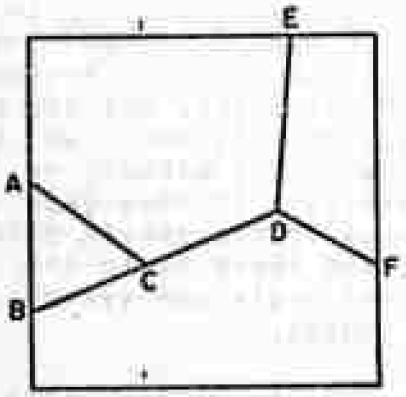


Fig. 10.

EXAMPLE OF INFERENCE OF THE PRESENCE OF COMPLEX CORNERS

the perimeter of the window, but big enough so that we will have enough boundary points to get a good line fit. Also we want the size of the window to be small enough so that the assumption of almost uniform intensity inside the window is justified.

In our system, when we want to read in a window, we have to wait up to 1/30 sec. till this window is scanned again. Hence we might save time by taking in the part of the frame we want to search all at one time and then run on the stored image. There are two disadvantages: First in memory size and the number of memory cycles, and second, we lose the flexibility of choosing the clip levels for each window independently.

3.2 Program structure

3.2.1 General notes

We start with the description of FINDIMAG because its details are needed to understand the description of SRCHIMAG which calls FINDIMAG at various stages. However, FINDIMAG is not directly callable by the user (or other modules).

3.2.2 Structure of FINDIMAG

The block diagram of FINDIMAG is shown in Figure 11. The names of the blocks are by no means exact descriptions of them. Not all the blocks, and not only blocks are distinct procedures in the programming language sense. The descriptions of the blocks follow:

OPENCLIP
The clip levels are set so that the maximum dynamic range is digitized.

INPUT
All the parameters (camera number, window location, window size and clip levels) are checked for their legality and then the window is read once into store. If any of the parameters is illegal INPUT exits.

HISTO
The intensity histogram i.e for each intensity level (0 to 15), the number of samples within the window which have this intensity is found.

SETCLIP
The clip levels are set so that the dynamic range brackets the actual range of intensity levels found in

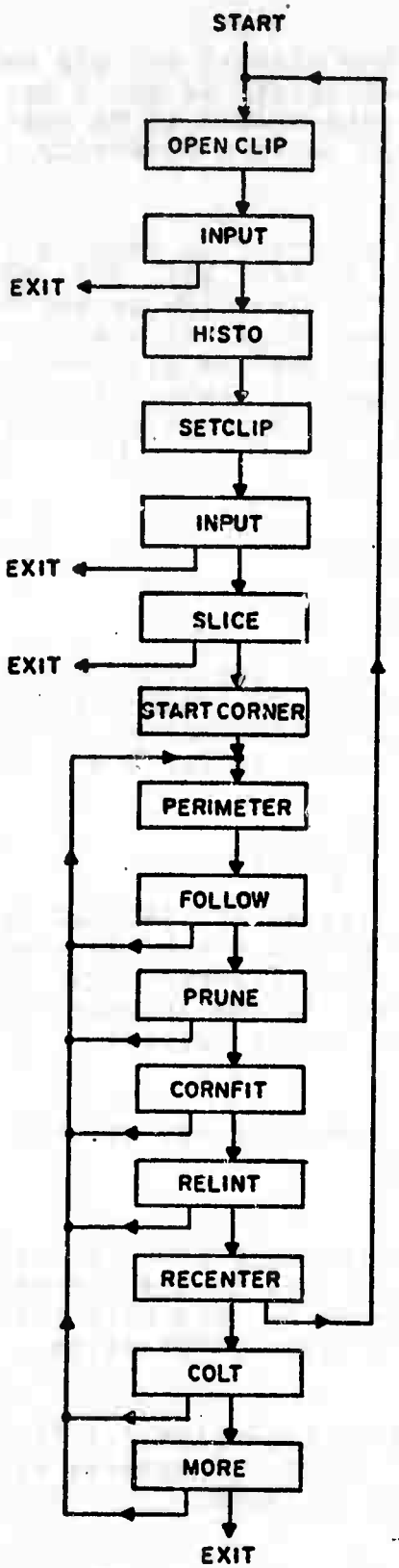


Fig. 11.

BLOCK-DIAGRAM OF FINDIMAG

HISTO. (See Figures 3 and 12). This insures that the full dynamic range of the A/D converter is utilized without losing any information. This is important since the whole operation of the corner-finder is based on finding two or more maxima in the histogram.

The computation of the new clip levels is as follows: If MAXI and MINI denote the maximum and minimum intensities in the window then

TCLIP=(15-MAXI)/2 and BCLIP=(15-MINI)/2.

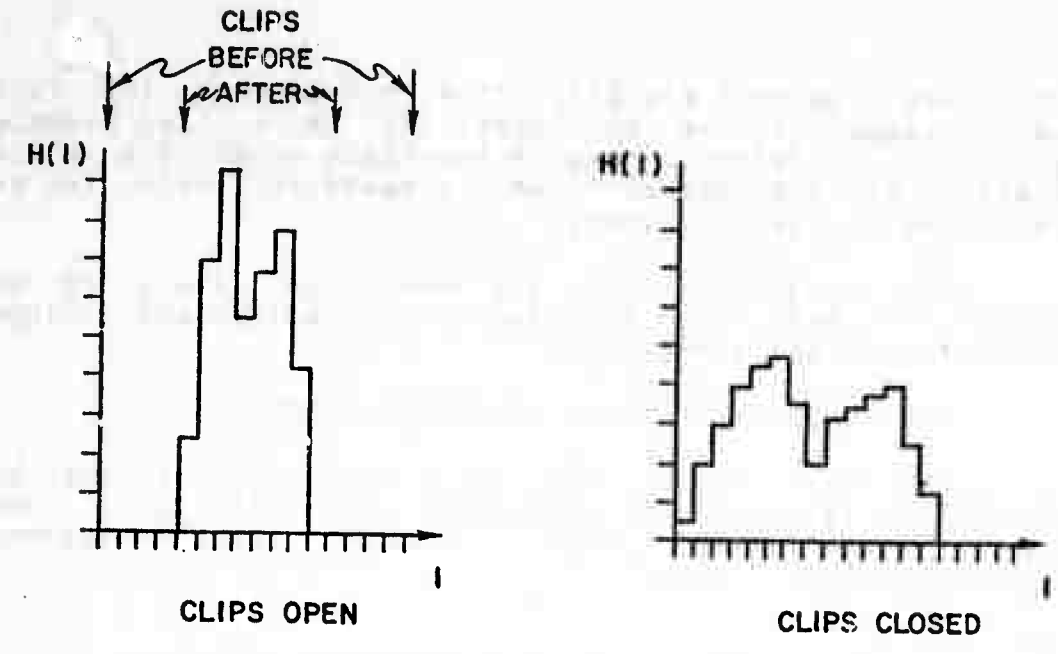
The clip levels can be closed even further by snipping the tails off the histogram. This will utilize the dynamic range even better without losing important information. (See Figure 13).

SLICE

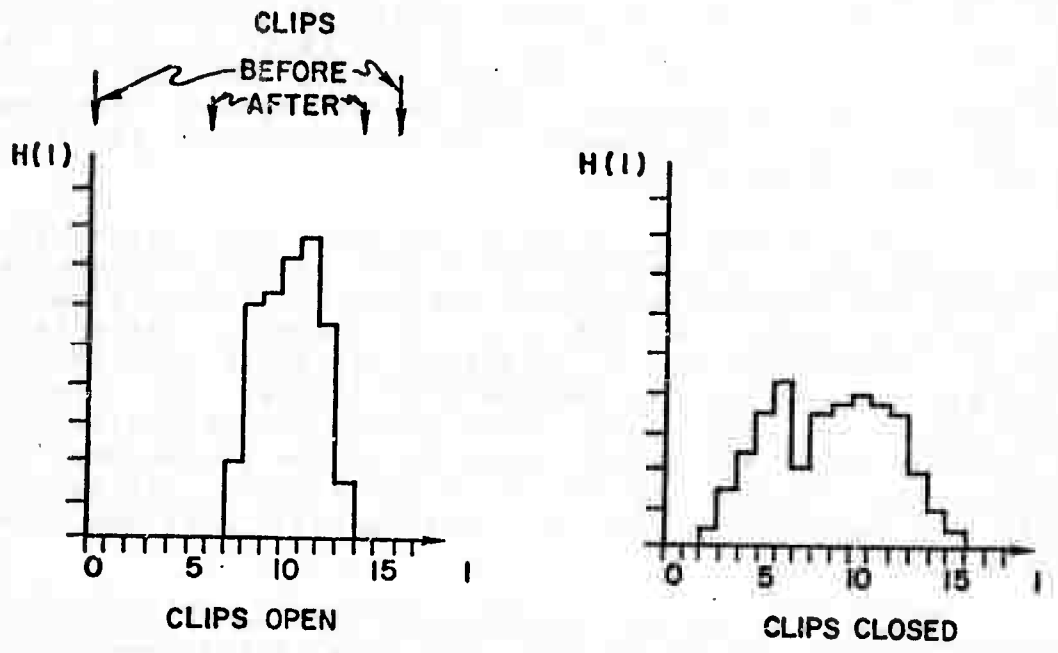
HISTO is used to form the intensity histogram (which we will call H(I)). Then the maxima of the histogram are found as follows: There is a maximum at intensity I if H(I) is larger than a given threshold, H(I)≥H(I-1) and H(I)>H(I+1). (H(16) is artificially made equal to 0).(See Figure 14). The threshold is empirically fixed at 1/16 of the total number of samples in the largest square inscribed in the window in accordance with the assumption about the corner image relative size.

After finding two or more maxima, the intensities which bound the intensity spread in each region are computed by taking the mid-points between the maxima. Intensities -1 and 16 are added to the list of bounds. There is an option, determined by a parameter given to FINDIMAG, to merge all the regions but the one with highest (or lowest) intensity, into one region. This option can save time when it is known that the inside intensity of the corner has the highest (or lowest) intensity in the window. In certain cases this serves also as an "anti-shadow" device. (See Figure 15).

There are two exit conditions from SLICE which will cause FINDIMAG to terminate. The first occurs when it is discovered that SETCLIP set the clips at the same value, which means that the intensity is almost uniform over the window. There is no point computing the histogram in this case since the noise will make it unreliable. The other condition occurs when it is discovered that the histogram has only one maximum. Note that in this case the spread of intensities is larger than in the previous case which means that there might be actually two regions in the window. (See cases C & D in Figure 8). The reason for losing one of the maxima in cases C & D is that one or both regions have



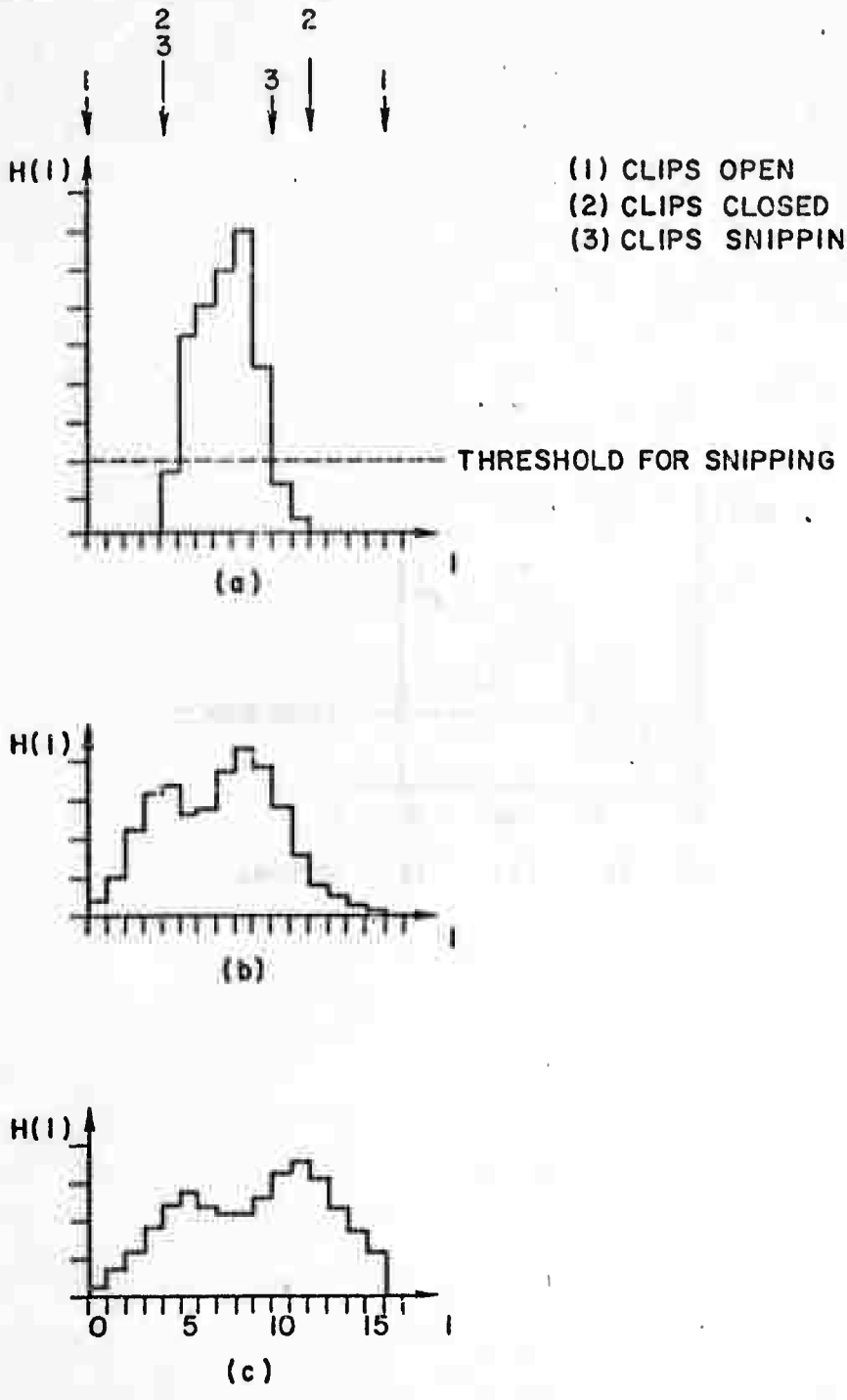
CASE A



CASE B

Fig. 12.

OPERATION OF SETCLIP



(a) Histogram with open clips.

(b) Histogram with closed clips.

(c) Histogram with snipping clips.

Fig. 13.

EXAMPLE OF THE EFFECT OF SNIPPING CLIPS

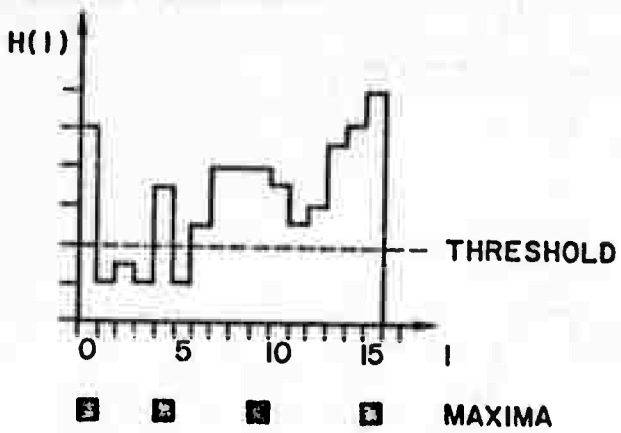
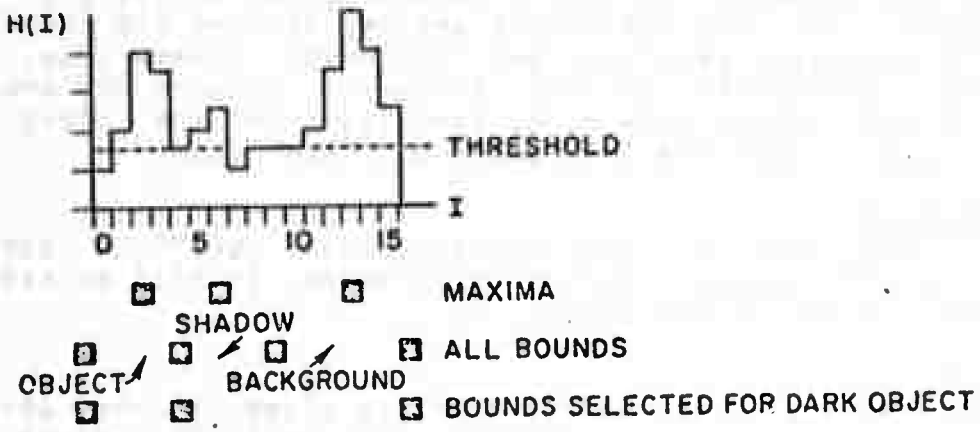


Fig. 14.
EXAMPLE OF MAXIMA SELECTION IN SLICE



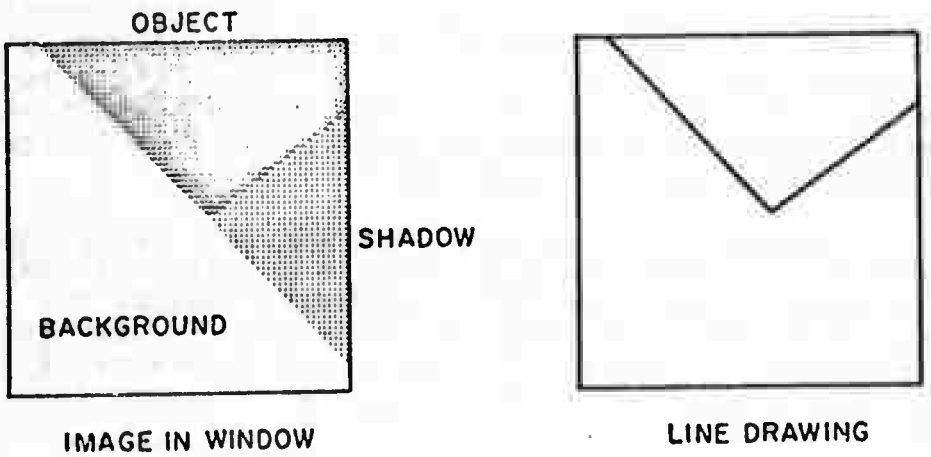


Fig. 15.

EXAMPLE OF BOUNDS SELECTION IN SLICE WHICH IGNORES THE SHADOW

wider spread of intensities than in cases A & B, which violate one of the basic assumptions.

If it is decided to try harder than the current version of the corner-finder, we can go in either or both of the following ways: (a) Make a more detailed analysis of the histogram. (b) Try again with smaller windows and check if their histograms peak at two different intensities. To compare intensities of two different windows the analog video signal values have to be computed from the digitized video signal and the clip setting in each case.

STARTCORNER

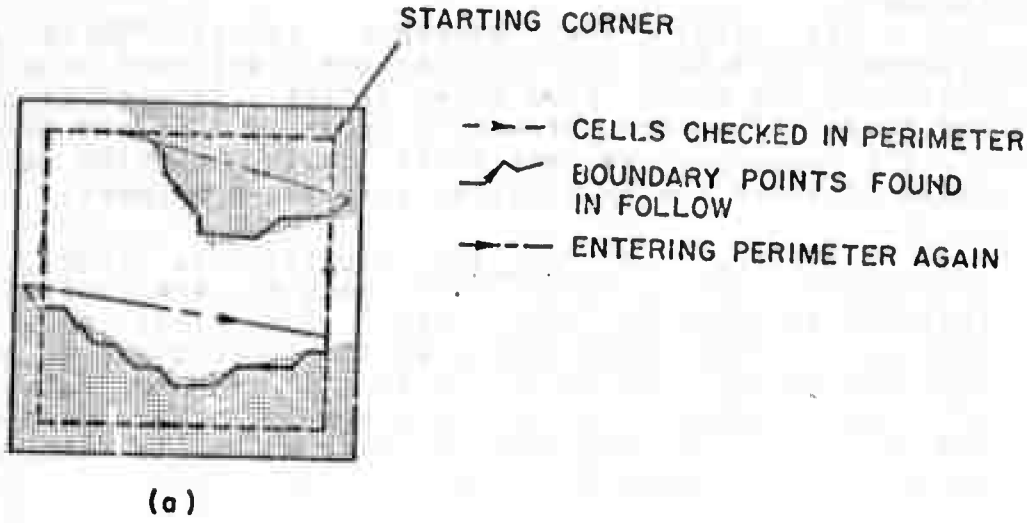
The corner of the window where PERIMETER starts checking for boundary points is selected. It will be easier to describe this block later.

PERIMETER

Using the assumptions that the sides of the corner intersect the window perimeter and that the sides are actually the boundaries between regions, the points along the perimeter are checked in CCW order starting at a corner of the window chosen in STARTCORNER. Using the bounds computed in SLICE, the region which the starting point belongs to is established and consecutive points along the perimeter are checked to see if they belong to the same region. If the next point belongs to a different region the block FOLLOW is entered and the boundary is followed until it intersects the perimeter again.

After the boundary information is processed and there is a need to find another boundary, either because this boundary was rejected or because FINDIMAG was told to find more than one feature inside the window, PERIMETER is entered again. The starting point now is the point next to the one where the perimeter was left last to enter FOLLOW. (See Figure 16(a)).

In order not to trace each boundary twice, once coming from one region and then again coming from the other region, FOLLOW is entered only if the next point belongs to a region of higher intensity in the case that we are looking for a corner with low inside intensity, or by convention when using all the bounds, and vice-versa when we are looking for high inside intensity. These particular directions are picked because it is assumed that the background is more "noisy" than the object's actual intensity, and hence we will get a smoother boundary by tracing it from the inside. When all bounds are used, the convention mentioned above will in some cases cause the loss of a part of a complex corner.(See Figure 16(b)). This part can be recovered, however, by analyzing the same window



(a) Example of the order in which cells are checked and boundary points are found (all bounds are used).

(b) Example of an undetected edge (all bounds are used).

Fig. 16.

OPERATION OF PERIMETER

again, this time looking for a corner with high intensity.

FOLLOW

To find the boundary points FOLLOW uses a maze-solving trick: walking with the right hand continuously touching the wall. Some added trickery is used to minimize the number of points checked in order to find the next point of the boundary as explained below. No point is checked twice if there are no loops or very sharp corners.

After a point on the boundary is found, its 8 neighbours are checked (the samples are spaced on a rectangular grid) in CCW order. The first point which belongs in the same region as the current one is the next boundary point. The direction of the neighbour point which will be checked first is the diagonal in CW direction from the direction from the last boundary point to the present point. (See Figures 17(a) and 17(b)).

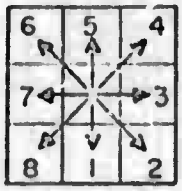
FOLLOW will automatically enter PERIMETER Again in the following cases:

a) The first point checked belongs to a region different from those of all its 8 neighbours. Such a point is called a singular point. (See Figure 17(c)).

(b) The boundary loops on itself and intersects the perimeter again exactly at its first point. This kind of looping is different from local loops which are pruned in the next block. (See Figure 17(d)).

(c) The number of points found on the boundary is smaller than a threshold empirically fixed at half the smaller dimension of the window, in accordance with the assumption that the image of the corner occupy more than a certain part of the window area. Ideally this region should be ruled out by the threshold in SLICE but there are cases when we have in one region a small patch of intensities in the same interval as another region.(See Figure 17(e)). Note that the above threshold corresponds to a square region whose area is the threshold in SLICE. Any region with the same area which "invades" the window more will have a longer boundary and thus would not be rejected by FOLLOW.

FOLLOW uses two kinds of data structures in its operation. One is a rectangular array with the dimensions of the window. When a point at window coordinates I,J is found as a boundary point, a 1 is stored in cell I,J in the array, so that if a point is found again a local loop is indicated. The other structure is a linear array in which the coordinates of a boundary point and the above number is

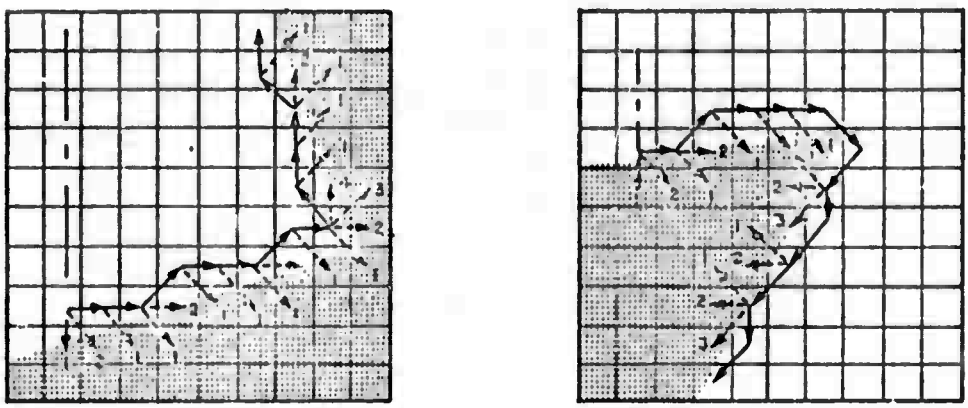


D1	1	2	3	4	5	6	7	8
D2	8	8	8	2	4	4	6	6

D1 : DIRECTION FROM LAST BOUNDARY POINT TO PRESENT POINT

D2 : DIRECTION FROM PRESENT BOUNDARY POINT TO NEXT POINT TO BE CHECKED

(a)



(b)

---→ POINTS CHECKED (IN ORDER OF CHECKING)

——→ BOUNDARY POINTS FOUND

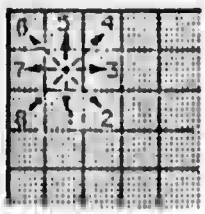
(a) Order of points checked in FOLLOW.

(b) Examples of "maze-walking".

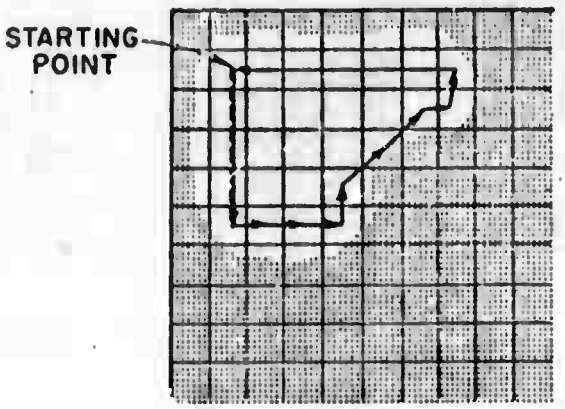
Fig. 17.

OPERATION OF FOLLOW

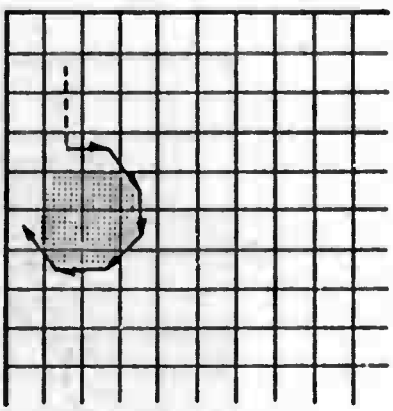
(Continued on next page)



(c)



(d)



(e)

(c) Singular point.

(d) Loop.

(e) Patch.

Fig. 17.

OPERATION OF FOLLOW
(Continued)

stored in the order in which the points were found. The use of two diferent data structures to store the same information is of course redundant but it saves processing time. Instead of using the rectangular array (which can be 88k words long if a fullsize window is ever used) the linear array can be searched to find out if a boundary point has been already found once, or a hash coding scheme could be used since the rectangular array is sparse. The general problem of memory size needed vs. processing time is discussed in Chapter 7. Every time that PERIMETER is entered, the rectangular array is zeroed and the linear array pointer is reset.

PRUNE

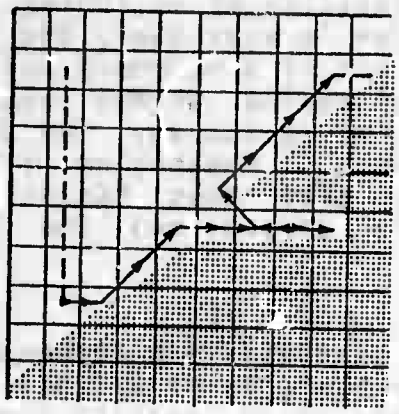
In PRUNE local loops and branches are pruned from the boundary points linear array by deleting all the points between two occurrences of the same point, merging these two occurrences and compacting the array. (See Figure 18).

PRUNE automatically enters PERIMETER again if the number of boundary points left after pruning is less than the threshold used in FOLLOW.

CORNFIT

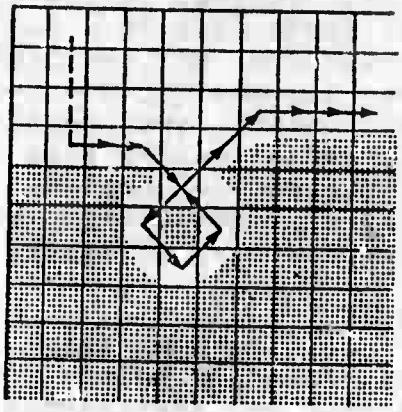
CORNFIT tries first to fit one straight line to the set of boundary points. The fit is a least square normal (i.e perpendicular to the proposed line) distance fit. If the average (per point) distance is more than a threshold, empirically set at .25 raster units, the line is rejected and a corner is tried. The set of boundary points is broken into two sets after each boundary point in order, starting after the third point (empirically chosen) and ending before the third from last point. A straight line is fitted to each subset in the same sense as above. Some processing time is saved by updating the sums and sums of squares instead of computing them each time the break is moved, which is actually moving a point from one subset to the other. The break which yields the minimal average error is chosen and the average error is compared to a threshold empirically set at .5 raster units. If the average error is larger than the threshold, PERIMETER is entered again, otherwise the intersection of the two lines is computed. If the intersection point lies outside the window, the corner is rejected and PERIMETER is entered. This can happen, for example, if two lines which do not intersect inside the window but come close together are accidentely connected by a noisy point or patch, or if there is a line which is very noisy. (See Figure 19).

The information about the corner is stored in the first 7 of an 8 cell linear array in the following order: direction cosines of the first side (see Section 3.1.2 for



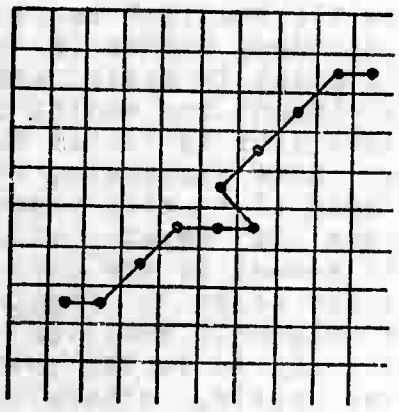
(a)

Creation of a branch by FOLLOW



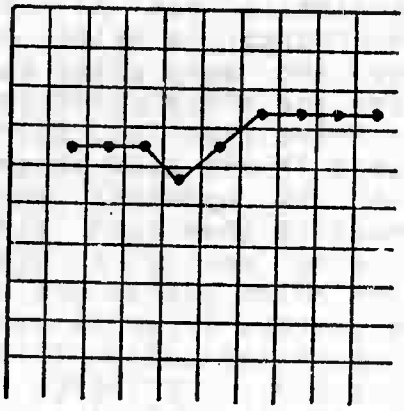
(b)

Creation of a loop by FOLLOW



(c)

The branch in (a) was pruned

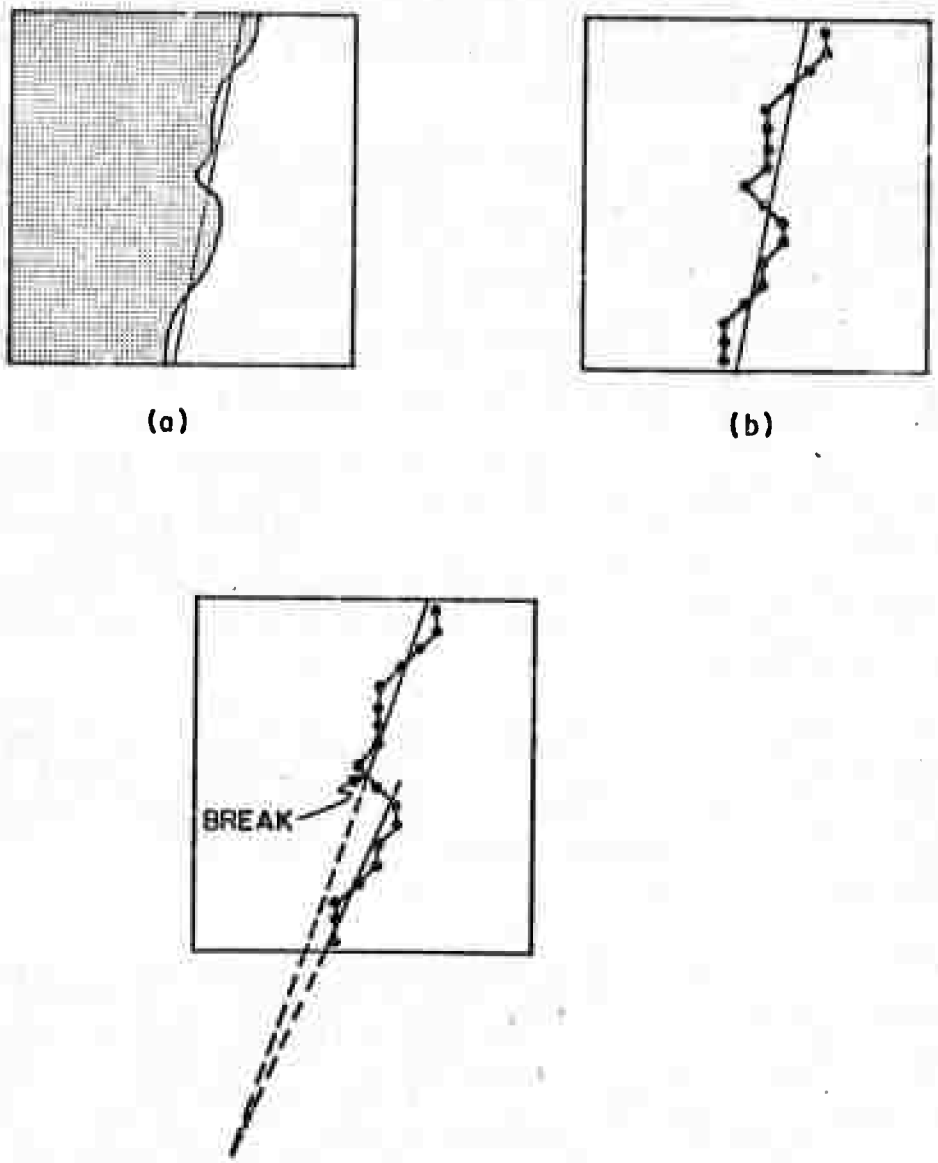


(d)

The loop in (b) was pruned

Fig. 18.

OPERATION OF PRUNE



(a) (b)

(c)

(a) Original line and noise.

(b) Boundary points and fitted line (rejected).

(c) Boundary points and fitted corner (also rejected).

Fig. 19.

EXAMPLE OF A VERTEX OUTSIDE THE WINDOW

definitions), direction cosines of the second side, location of the vertex (in case of a line it is the midpoint), all in image coordinate system, and whether a line or a corner was found (1 or 0 accordingly).

A line is tried before a corner because if the boundary is really a line, a corner would fit it at least as well as a line (i.e with smaller average error). The vertex in this case would be in an arbitrary location along the line. However the information that the border is indeed a line is important and should be preserved. Alternately a corner could be fitted first and then a line should be tried only if the inside angle of the corner is close to π or one of the sides is much shorter than the other.

We mentioned already that a patch on the perimeter will be rejected because its outline is not long enough. If the patch is on the boundary it will either be pruned out as a branch or a loop, or it will slightly perturb the line fitted to the boundary points. A patch in the middle of a region would not be seen at all, although it could affect the intensity histogram.

The situations that can cause failure although two peaks have been found in the histogram are: (a) "Salt and pepper" texture, (b) A boundary containing more than one corner inside the window. Note that both these cases contradict the basic assumptions so that failure is not surprising.

RELINT

If the boundary points were fitted by a corner in CORNFIT and if a relative intensity match is demanded, it is done in this block. To find the relative inside intensity, RELINT uses the following fact: If the two highest bounds are used, and if the first side was fitted by the boundary points found first in FOLLOW, then the inside intensity is high. This situation is caused by the definition of corner sides and by the conventions established in PERIMETER. Similar considerations are made for the other combinations of bounds used and inside relative intensity. (See Figures 16 and 17 for examples).

If the relative intensity is checked and a match is not achieved, PERIMETER is entered again.

RECENTER

If the location of the corner or the midpoint of a line is not close enough to the center of the window (the threshold is empirically set at 1/8 of the dimension of the window in each direction), and if recentering is permitted (as determined by one of the parameters), the window is

moved so that its center will coincide with the vertex (or midpoint) and the complete process is started again. This is done partly because if the corner is not centered, one of the sides may be short and the line fitted to it inaccurate. Other reasons will become apparent in the discussion of SRCHIMAG.

COLT

In this block the form of the feature found is matched if a match is demanded. The form of the feature to match against is given in the same data structure that is used in CORNFIT. A match is achieved if the cosine of the angle between the two sides to be matched is closer to 1 than a threshold given by a parameter. The cosine of the angle is chosen as a criterion since it is easily computed from the direction cosines stored in the data structure and is invariant to angle direction.

Three kinds of matches can be demanded, determined by a number stored in the 8th cell of the given feature data structure:

(a) Both sides have to be matched.
(b) At least one of the found sides has to match the first (second) side of the given feature.
(c) At least one of the found sides has to match either of the sides of the given feature.

If there is no match in the last two cases PERIMETER is entered again. If there is no match in case (a) a contra-match is tried: we check to see if any of the sides found has a direction opposite (within the above tolerance) to either of the given feature sides. This information is needed to aid the search for the feature (see discussion of SRCHIMAG). If no contra-match is found, PERIMETER is entered again.

The output of COLT is stored in the 8th cell of the linear array mentioned already in CORNFIT as follows: -1 if no match was obtained, 0 if a match was obtained in (a), or the side that matched (1 or 2) in (b) and (c), or the side that contra-matched in (a).

STARTCORNER

PERIMETER begins to check the points at a corner of the window. If the form of the corner sought is known, i.e. stored in the data structure of the corner to match against, we can select a starting corner which will minimize the number of points checked in PERIMETER before FOLLOW is entered, in the following way: In most situations we would like to trace the boundary from the inside of the corner, because we assume that the background is more noisy.

Therefore we select the first corner of the window in a clockwise direction from all possible positions of the first side of the given corner, as the starting point for PERIMETER. (See Figure 20).

MORE

MORE will exit under any of the following conditions:

(a) The number of corners (or lines) that FINDIMAG was told to find, have been found and properly matched.
(b) A corner is sought, there is no demand to match form, searching is permitted and a line was found.
(c) A feature is sought while searching is permitted and a corner or a line was found which contra-matched the given feature form.

Otherwise PERIMETER is entered again.

3.2.3 STRUCTURE OF SRCHIMAGE

The block diagram of SRCHIMAG is shown in Figure 21. The description of the blocks follows.

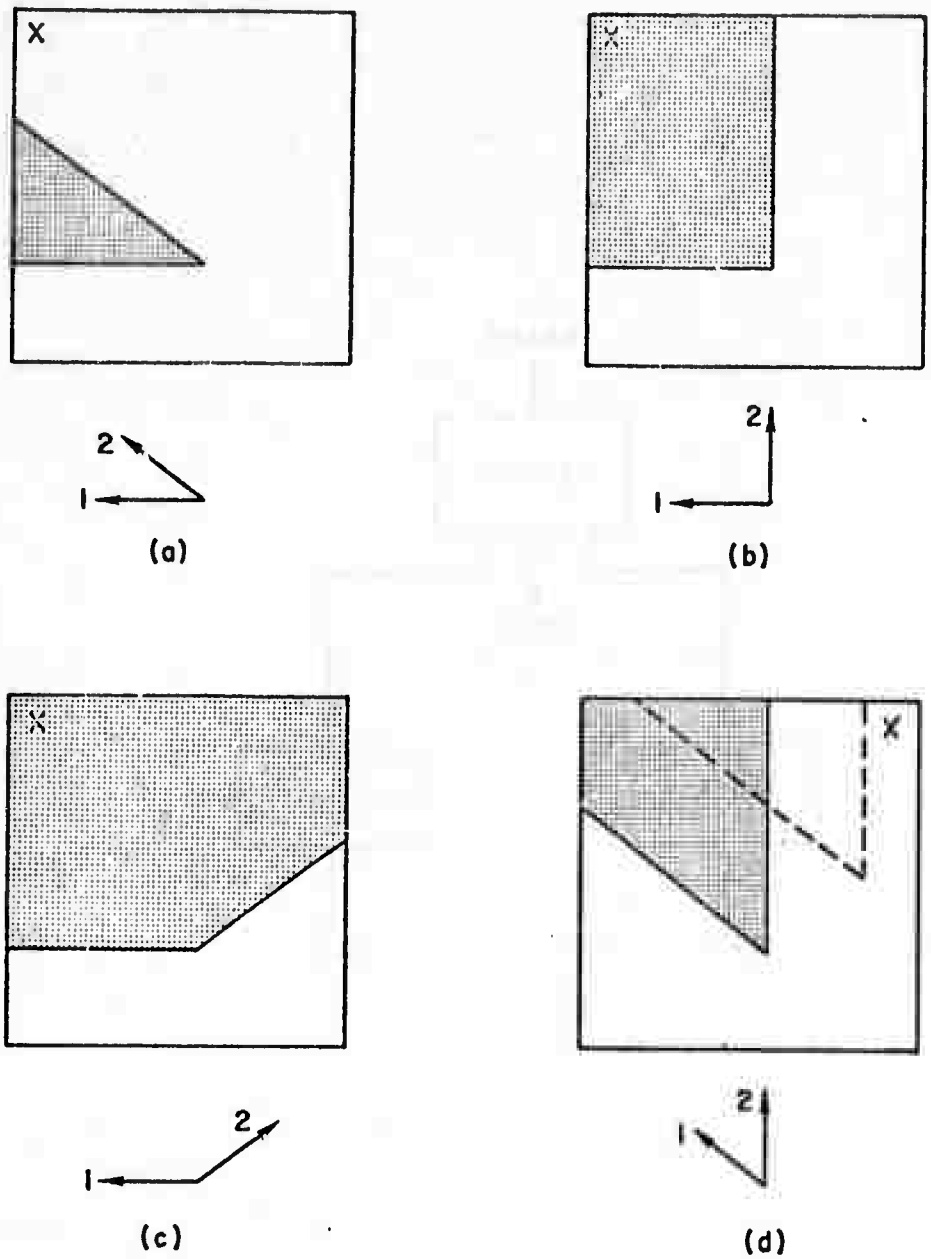
SEARCH

If search is not permitted, FINDIMAG is entered and then SEARCH exits. If search is permitted, it is done in a spiral scan pattern. The first position of the analyzed window is chosen to be at the most probable place to find the feature as determined by information which SEARCH has access to. From this position the search spirals outwards with overlapping, (see Figure 22). At each position FINDIMAG is called. The spiral search is continued until one of the following conditions is satisfied:

(a) The corner or line searched for is found and properly matched. In this case SEARCH exits.
(b) A feature is searched for with form match and a corner or a line is found which contra-matched the given feature. In this case FOLMATCH is entered.
(c) A corner is searched for, no form match is needed and a line was found. In this case FOLINE is entered.
(d) The next window position is partly outside the permitted search space. In this case SEARCH exits and a failure is indicated. The search space is a rectangular part of the frame. Its center coincides with the first window center and its size is determined by a parameter given to the module.

FOLINE

The position of the window when FOLINE is entered is stored. Then the window center is moved along the direction



X - Location picked for the starting point of PERIMETER.

Fig. 20.

EXAMPLES OF THE OPERATION OF STARTCORNER

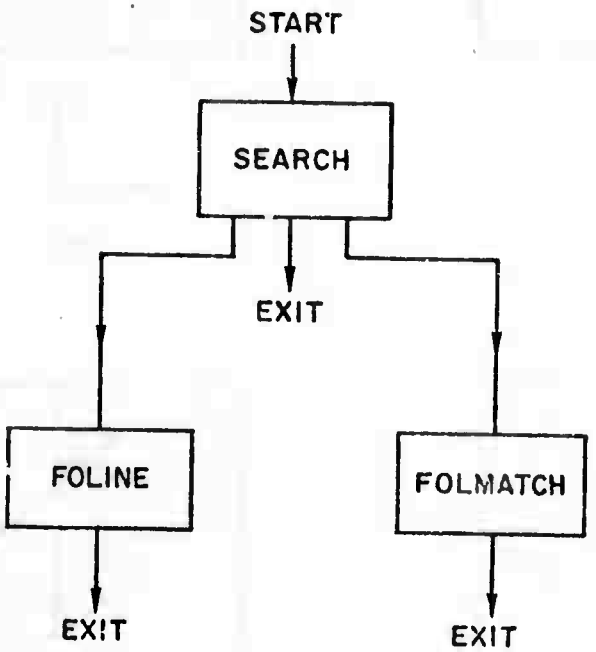
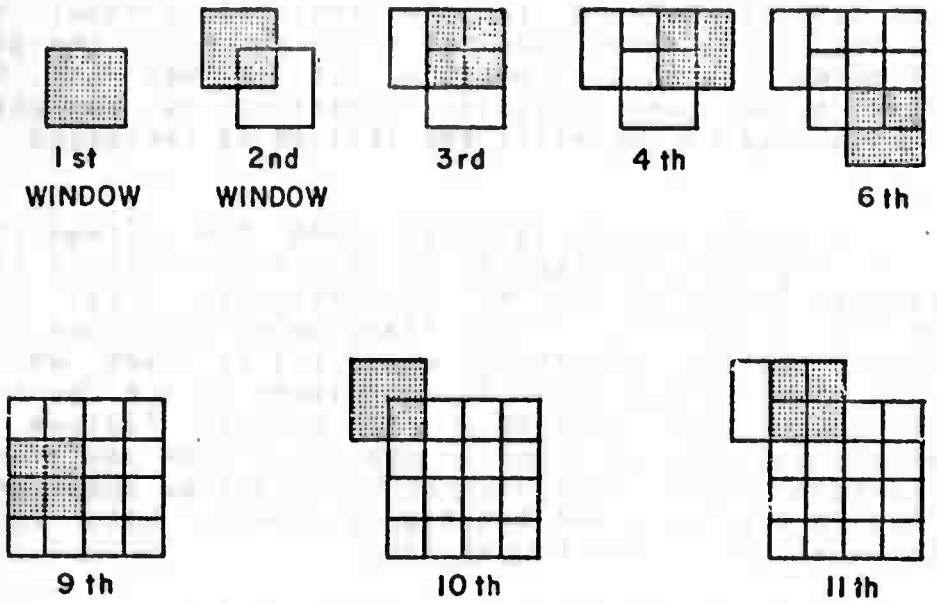
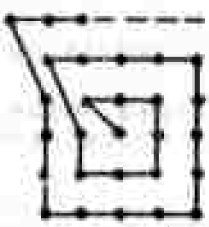


Fig. 21.

BLOCK-DIAGRAM OF SRCHIMAG





Window center locus plot
(Window size same as above)

Fig. 22

SPIRAL SEARCH IN SRCHIMAG

of the line by steps of .5*(window size). If a line is found again, the window continues to move along the line till it reaches the search space border. Then the line is followed in the other direction from the position stored when FOLINE was first entered till the search space border on the other side is reached. In this case FOLINE exits and failure is indicated.

If a corner which matches properly (not a form match) is found, FOLINE exits successfully. If at any position nothing is found in the analyzed window, or a corner was found which does not match properly, the search is abandoned in this direction. If it was the first direction tried, the opposite direction is searched as above. Otherwise FOLINE exits and failure is indicated.

FOLMATCH

The window center is moved along the direction of the side that contra-matched, by .5*(window size) if the side belongs to a line or by (2/3)*(window size) if it belongs to a corner. This difference assures that in the next window position the corner would not be seen at all, otherwise the window would be recentered on the corner and FOLMATCH would loop miserably. If another feature with contra-matching side is found in the new window position, it is followed as above. Otherwise FOLMATCH exits, successfully if a feature was found and matched properly, with failure indication if not. (See Figure 23).

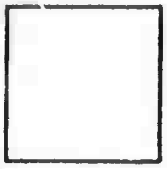
Figure 24 gives an example which shows why recentering is so important for searching when form match is needed, although it can cause looping if proper care is not taken.

3.3 User parameters and corner-finder module output

There are two kinds of parameters which the user has to supply to the corner finder, global and formal. The global parameters are:

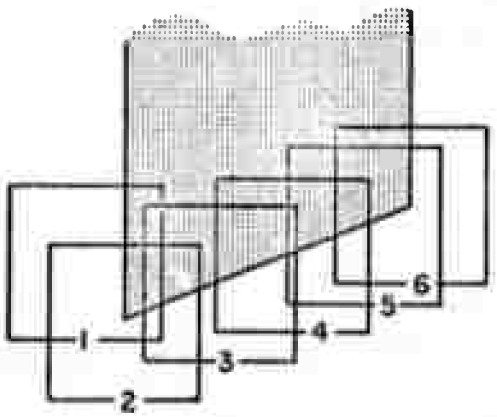
(a) The number of the camera to be used. Currently only one camera is used but shortly there will be two.
(b) Window specifications: the coordinates of the center in the l.c.s., width and height.

These parameters occupy in that order the first 5 of an 8 cell linear array (called LOOK_AT). The other 3 contain the current setting of the clips (BCLIP and TCLIP) and the target voltage. These 8 parameters are needed (in principle) to compute the light intenelty distribution on the portion of the vidicon face, corresponding to the window, from the intensities stored at the sampling points.



(a)
Window size

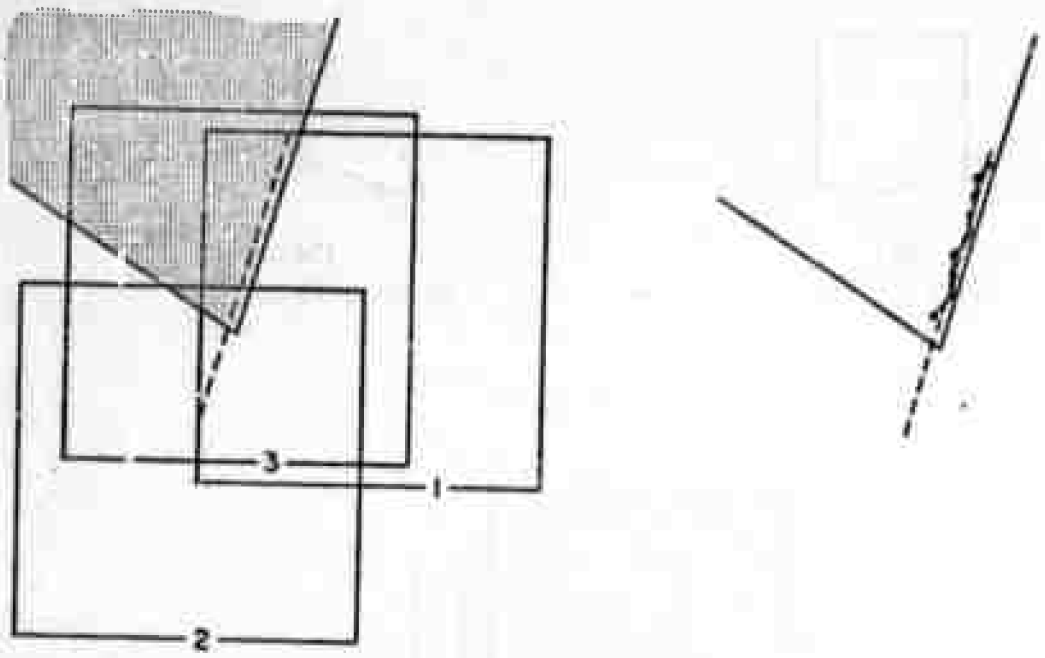
(b)
Given corner form



(c)
Sequence of window positions

(1) Corner (wrong one) found.
(2) Recentering.
(3) Side 2 contra-matched and followed.
(4) Line followed.
(5) Corner found.
(6) Recentering.

Fig. 23.
EXAMPLE OF CONTRA-MATCH FOLLOWING



(1) Original window and fitted line.

(2) Contra-match followed without recentering. Not enough boundary points found.

(3) Window recentered and the corner found.

Fig. 24.

EXAMPLE OF THE NEED TO RECENTER

For those who will actually use the corner-finder programs I will use the names of the formal parameters as they appear in the program. The names do not necessarily express their meaning. They are:

(a) BKGR (see disscussion of SLICE in Section 3.2.2). BKG_R can have three values: -1 when we want to use the two highest bounds, for example in the case that we know that the corner is relatively light; 1 when we want to use the two lowest bounds, for example in the case that we know that the corner is relatively dark; 0 when we want to use all the bounds, for example in the case that the relative intensity is not known or that we are looking for more than one feature.

(b) INT (see discussion of RELINT in Section 3.2.2). INT can have three values: -1 when we want to match relative intensity and we are looking for lighter inside; 1 when we want to match relative intensity and we are looking for darker inside; 0 when we do not demand a match of relative intensity. BKGR and INT do not necessarily have the same value although they usually do. For example if we want to trace a dark corner from the outside but match its relative intensity, and it is expected that we will have only two regions in the window, then we will set BKGR to -1, but INT to 1.

(c) SEARCH (see discussion of RECENTER and MORE in Section 3.2.2 and of the block SEARCH in Section 3.2.3). If SEARCH>0, recentering and searching are permitted. The dimensions of the search space are then given by the value of SEARCH+1 times the dimensions of the window used. The number of windows needed by the spiral scan to cover the whole search space is: $(2*SEARCH+1)^2$. If SEARCH=0, recentering is permitted but not searching. If SEARCH<0 neither searching nor recentering is permitted. In this case -SEARCH is the number of features (corners and/or lines) wanted inthe window. Implicitly only one feature is sought if SEARCH≥0. (See discussion of simple and cluttered environments in Section 3.1.3).

(d) TOLER (see discussion of COLT in section 3.2.2). If TOLER<0 no match of form is demanded. If TOLER>0, match of form is demanded and the value of TOLER is the tolerance. Since the tolerance is compared with the difference between 1 and the cosine of the angle between the two sides to be matched, values of TOLER larger than 1 are equivalent to no demand for match. The form of the feature to match against is given in the next and last parameter.

(e) DIRC is an 8 cell linear array. The first 4

cells contain the direction cosines of the two sides of the given feature. The next 2 contain predicted coordinates of the vertex of the corner. They are not used in the matching process. The 7th cell determines if a line (1) or a corner (0) is sought. The 8th cell determines whether we demand a match of both sides (0), first side only (1), second side linear arrays, one for each feature if SEARCH<0, described

When the corner-finder is working under the hand-eye monitor it is activated by the follwing message procedure:

SRCH-IMAG(BKGR,INT,SEARCH,TOLER,DIRD);
In this case the following outputs reside in global cells.

The output of the corner-finder is a set of 8 cell linear arrays, one for each feature if SEARCH<0, described in CORNFIT and COLT in Section 3.2.2. In MORE it is decided if a feature, which is stored in the last array filled, satisfies all the demands, otherwise this array is erased from the set. If the corner-finder failed, the reason for failure is stored (as a number) in a special cell (called EYEFLG). For debugging purposes many intermediate results are displayed during program execution. (See Figure 25 for an example).

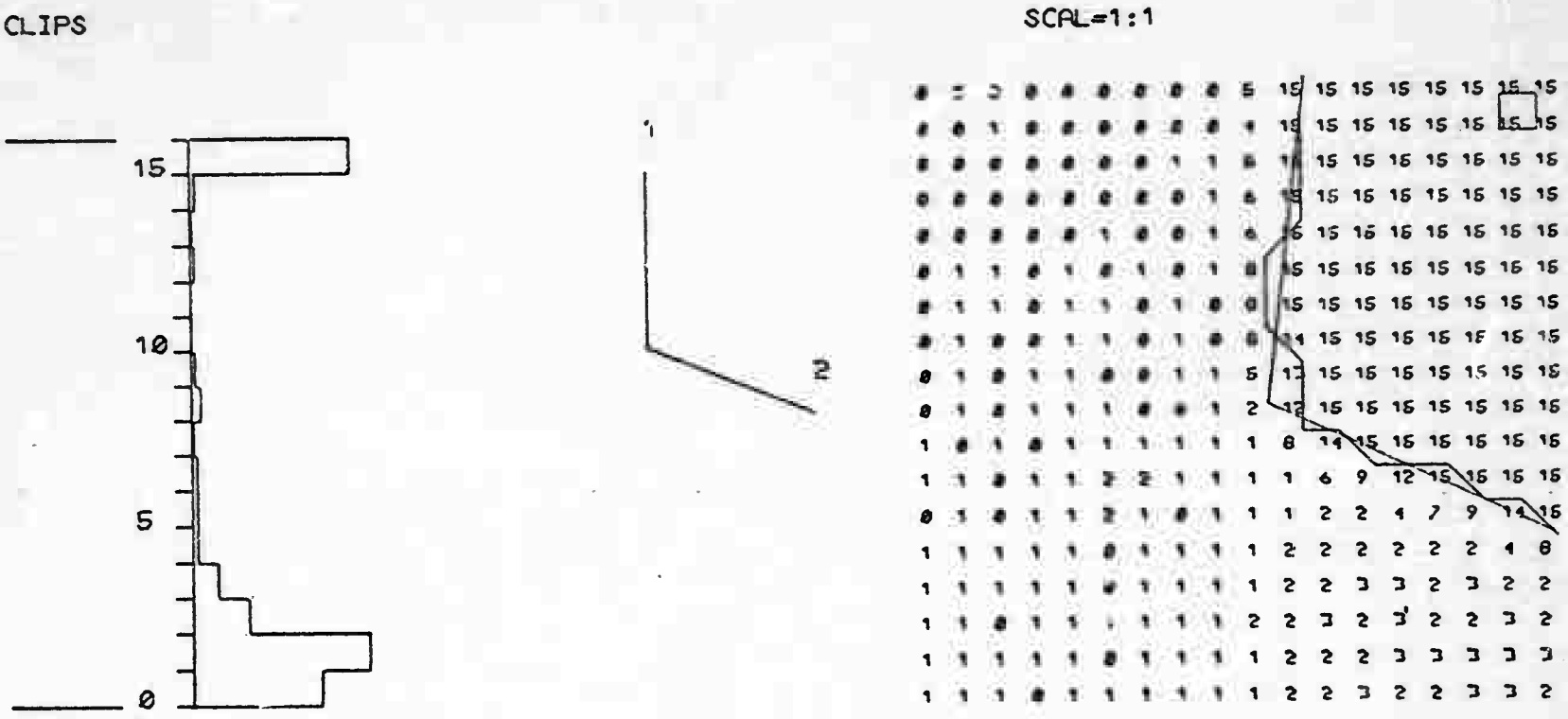


Fig. 25.

CORNER-FINDER MODULE RESULTS DISPLAY

CHAPTER 4: CALIBRATION UPDATING

4.1 Introduction

4.1.1 General notes

The purpose of the calibration updating programs is to maintain the accuracy of the camera model against changes over time.

We start the discussion by describing in detail the general causes of the inaccuracies of the hand-eye system, especially those affecting the precision of manipulations. Then we describe the camera model and its calibration programs as developed by I. Sobel [26]. Analysis of the causes of variations leads to the description of the updating of the pan tilt and focus parameters.

4.1.2 Hand eye coordination

Let us define a nominal coordinate system (c.s.) which we call the "table coordinate system" (t.c.s.), (see Figures 1 and 26), as an orthogonal right-handed three dimensional system with:

(a) The plane z=0 is "close" to the face of the table.

(b) The origin is "close" to the corner of the table near the arm.

(c) The x and y axes are "close" to the edges of the table. (x going upwards in Figure 1 and y to the left).

Each point in space can be uniquely defined by a triple of numbers which are its coordinates in the t.c.s. We (the users) do not have any means of finding this triple of numbers since the t.c.s. is hypothetical. We can get approximations by the use of some assumptions (for example, the planarity of the table's face) and some measuring devices like rulers, a right angle tripod and a grid paper glued on the table's face. We will call this approximation the "user-table coordinates" (u.t.c.) of the point.

The hand-eye system has two ways to find approximations to the coordinates of a point in the t.c.s.: One is by using the camera(s) and the other by using the arm.

First we will discuss the determination of 3-D coordinates using the camera(s): The use of one view from one camera, is inherently not enough to determine 3-D coordinates. This problem is discussed extensively elsewhere [26]. I would like however to comment on four possible solutions:

(a) Stereo- i.e. the use of two views obtained with one or two cameras from two points which are not colinear with the point to be measured. The main problems with this solution are accuracy for "narrow angle" stereo and correspondence for "wide angle" stereo [21].

(b) Focus information- The automatic focussing module can be used here. Its accuracy can be estimated by the minimal depth of field information given in Appendix B. (.5 to 1 inch) which is not enough. See also [11].

(c) Stadiametric information- i.e. the use of known dimensions of objects. This method is comparable in accuracy to that mentioned above for the automatic focussing program. The accuracy here is limited by the accuracy of the scene analyzer and inherently by the resolution of the sensor.

(d) Support hypothesis- i.e. supplying external information that the point lies on a particular plane or line specified in the t.c.s. This method is the one used extensively in the hand-eye system in ways described in this report, (see also [22] and [8]).

A fifth, somewhat unrelated, method uses controlled illumination of the scene. This method was implemented by Agin at Stanford, using a planar laser beam to scan the scene[1].

The most common application of the support hypothesis is when a point lies on the surface of the table which we assume is the plane z=0. The approximation computed by the camera using the support hypothesis we will call the "camera-table coordinates" (c.t.c.) of a point. The difference between the coordinates of a point in the t.c.s. (which we can not measure) and its c.t.c. have the following causes:

(a) the quality of the assumption that the table surface is the plane z=0,
(b) the quality of the image,
(c) the precision of the scene analyzing routine,
(d) the quality of the mathematical model of the camera,
(e) the precision of the invariant parameters of the model, and
(f) the quality of the mesurement of the variable parameters of the model.

(a),(c),(d) and (e) contribute the deterministic component of the difference. (b) and (f) contribute the random part of the difference. When we approximate differences of coordinates of two close points, using their

images in the same frame, the main contributions to the error are (b), (c) and parts of (d) and (e). The purpose of the camera calibration program developed by I. Sobel is to reduce this difference by improving the precision of the invariant parameters of the model. Some of the parameters assumed invariant are varying over time. One of the purposes of the calibration updating progams is to "track" these variations. The remaining variations add to the random component of the difference.

We now proceed to describe the determination of 3-D coordinates using the arm. When the arm control program is given a command to place the arm at some position, it tries to put the midpoint between the tips of the touch sensors at this position. Using the arm model and the reading of the potentiometers on the arm joints, we can compute an approximation for the coordinates in the t.c.s. of any point whose position relative to the hand is known. For example, if we assume that we know the exact opening between the fingers and the thickness of the rubber pad of the touch sensors, among other dimensions, we can compute the location of the hand-mark. We will call this approximation "hand-table coordinates" (h.t.c.).

Note that when a cube of known dimensions is grasped by the hand, we know the relation of its corners to the reference point of the arm only in the direction of the sweeping motion of the fingers (perpendicular to the finger face) but not in the other two directions. Hence the arm can not be used as a general measuring device, and we use the approximation given by the c.t.c. as our standard.

The causes of the differences between the coordinates of a point in the t.c.s. in its h.t.c. are similar to causes (d) through (f) given for the camera. The arm calibration program tries to reduce this difference by improving the values of the invariant parameters of the arm model.

The differences between the c.t.c. of a point and its h.t.c. have much more importance to us than the differences between either of them and the coordinates of a point in the nominal t.c.s. (or its u.t.c.), since the difference between the c.t.c. and the h.t.c. directly affect the precision of manipulation. We will call this difference, "coordination error". One of the purposes of the calibration updating programs is to reduce the coordination errors. In the visual feedback tasks the residual coordination error is measured in each situation and then reduced (below a given threshold) by incrementally specified movements of the arm.

From now on, when we mention coordinates in the t.c.s., we actually mean an approximation. Hopefully it will be clear from the context which approximation we mean (u.t.c., c.t.c. or h.t.c.).

4.2 Camera model and calibration

4.2.1 Camera model: general notes

The camera model has two parts which are called the external and the internal model. The external model deals with the parameters which affect the transformation of the external scene to an image formed on the vidicon face. The internal model describes the scanning and sampling processes by which this image is converted to readings at the sampling points. The model is geometric in nature and not photometric. The model allows us to compute for each point in the external world, specified for example by its 3 coordinates in the "table coordinate system" (t.c.s.), the sampling point in the i.c.s. which corresponds to each point and v.v. (in this case it will be a ray in the t.c.s. and not a point).

The geometric model and its calibration procedure were developed by I. Sobel [26]. A photometric model was developed by J. M. Tenenbaum [28].

4.2.2 Camera external model

The pan/tilt head and focussing mechanism are described in the model as follows (see Figure 26):

(a) The pan axis is perpendicular to the "table top" (this is the name given to the plane z=0 in the t.c.s.), and pierces it at point (P1,P2,0).

(b) The tilt axis is perpendicular to the pan axis and intersects it at point (P1,P2,P3).

(c) The location of the lens center and the direction of the principal ray is described as follows: we define a coordinate system which will be called the "camera coordinate system" (c.c.s.) so that its origin is at the lens center, its rotation relative to the t.c.s. is defined by the pan and tilt angles and a fixed rotation angle around the principal ray called SWING. The z axis is in the direction of the principal ray, the y axis is in a plane almost (because of SWING which is small) perpendicular to the table top and pointing towards the table top. The x axis completes the c.s. to be a r.h.s. The intersection of the pan and tilt axes is located at point (-DP1,-DP2,-DP3) in the c.c.s. Note that focussing is done by moving the vidicon

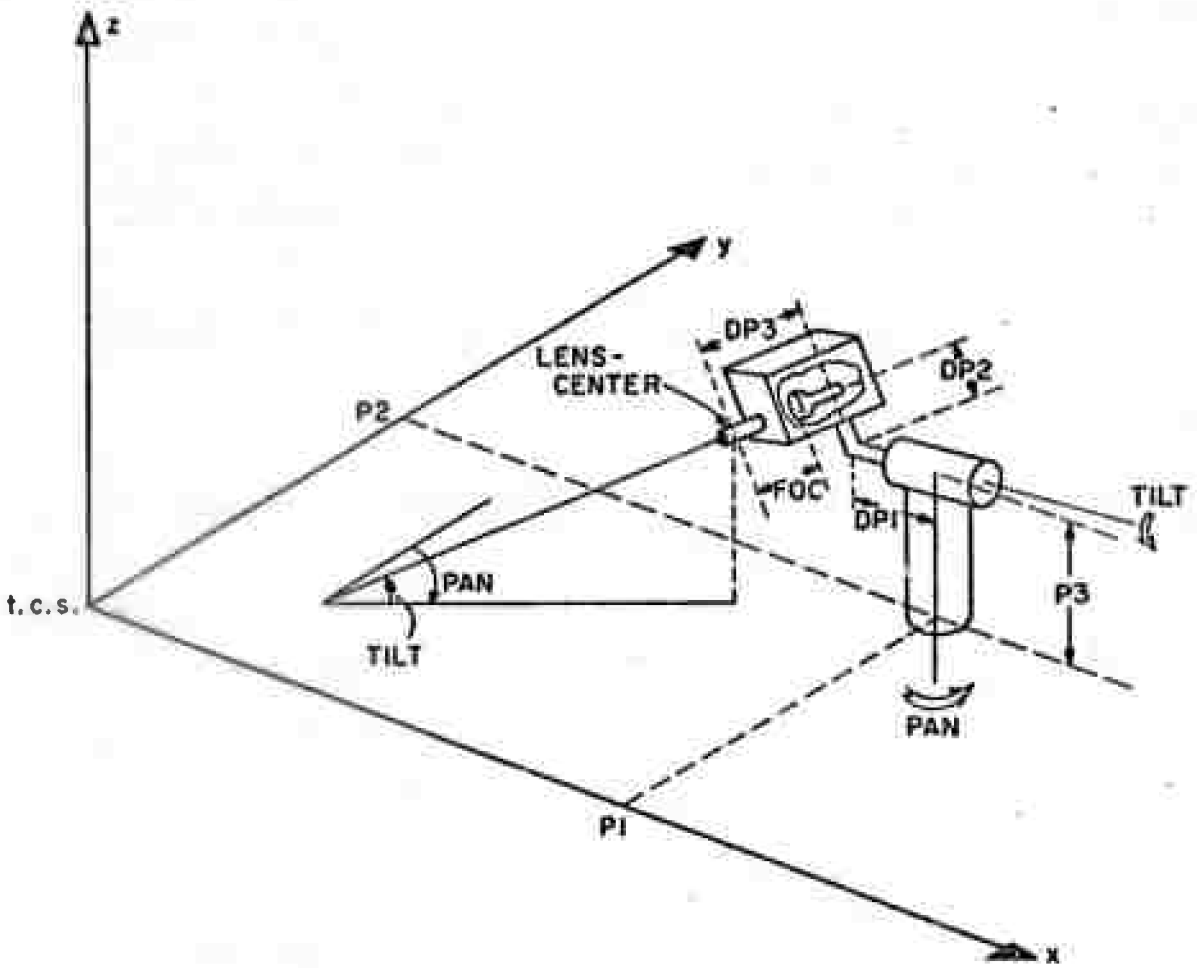


Fig. 26.

CAMERA EXTERNAL MODEL
(Drawn for SWING = 0)

and thus DP1,DP2,DP3 remain fixed for a given lens.

(d) The vidicon face is perpendicular to the principal ray and is located at a distance, which for convenience will be called "foc", from the lens center in the z-direction of the c.c.s. It is assumed that the optics can be described by the simple single lens geometric model with the focal length specified on the lens assembly.

(e) The pan and tilt angles and the distance foc can not be read directly by the computer. Instead, three potentiometers (pots) are used, two circular and one linear. They are connected to a regulated power supply. The voltages at their wipers are converted to digital form and stored when neccesary. We assume that the pot readings are linear with the pan and tilt angles and foc, and hence 6 parameters, scale and offset for each, are needed to convert the readings to pan tilt and foc. These parameters are called PPOTØ, PPOTD, TPOTØ, TPOTD, FOCLENØ, FOCLENG.

The camera external model is thus specified by 13 parameters: P1, P2, P3, SWING, DP1, DP2, DP3, PPOTØ, PPOTD, TPOTØ, TPOTD, FOCLENØ, FONLENG.

4.2.3 Camera internal model

The scanning and sampling processes are described in the model as follows (see Figure 27):

(a) The x and y axes of the i.c.s. are parallel to the x,y axes of the c.c.s.

(b) The principal ray pierces the image plane at point (PP1,PP2) of the i.c.s.

(c) The sampling resolution is KX samples/inch in the x direction of the i.c.s. (i.e. sampling along a scan line) and KY samples/inch in the y direction of the i.c.s. (i.e the number of lines per inch scanned). Usually KX differs from KY by only a few percent.

(d) The scanning starts at the origin of the i.c.s. The scan lines are almost parallel to the x-axis of the i.c.s., since each line ends where the next line starts. Hence the lines are at an angle of 1/333 rads. to the x-axis. Subsequent lines lie in the y-direction of the i.c.s.

(e) The effect of image inversion by the camera is cancelled by the fact that in reality the scan starts at the lower right hand corner of the frame and not at the i.c.s. origin.

The camera internal model is thus specified by 4 parameters: PP1,PP2,KX,KY.

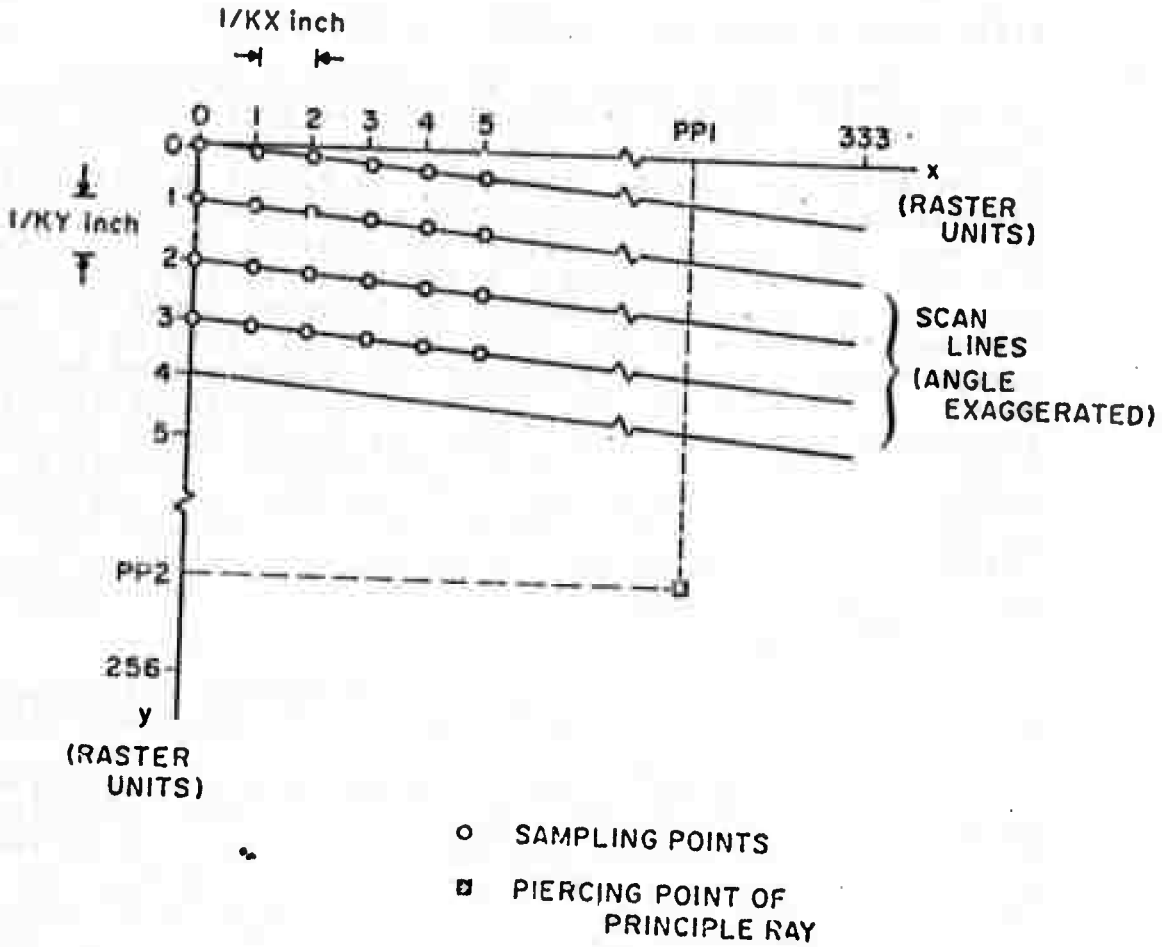


Fig. 27.

CAMERA INTERNAL MODEL

The computation of the transformations between the i.c.s. and the t.c.s. from the 17 parameters, the 3 pot readings and the lens focal length is explained in detail in [26] and is summarized in Appendix A.

Like the camera, the arm also has a mathematical model. This model has 32 parameters which are used, with the readings of pots in the arm joints, to compute the transformations from the t.c.s. to the "hand coordinate system" (h.c.s., a c.s. tied to the hand) and vice-versa.

4.2.4 Camera model calibration process

There is no easy way to generate a calibrated image on the vidicon face without using an external object (like a T.V chart) and the camera optics. Also there is no easy way to measure an image formed by the camera optics directly without using the vidicon and the scanning circuits. Hence we have no practical way of calibrating the external and internal models separately. A separate calibration can be done by a cut-and-try method as follows: Guess the values for the parameters of one model (say the external model), using these values, calibrate the parameters of the other model (the internal model). Then using the new values of the parameters of the first model calibrate again the second model and so on till some convergence condition is satisfied. This method was tried by Sobel and was found to be time consuming and ineffective. Although the time needed for each iteration is relatively short, (since a smaller number of parameters is handled), the number of iterations needed was large.

The calibration process developed by Sobel used the full (external and internal) model and is described below:

(a) An object (usually a black L shaped book end) of known size is put on the table at a known position. A picture of the object is stored with the readings of the pan tilt and focus pots and the location of the object vertices in the t.c.s. This process is repeated 10 to 20 times for various locations of the object on the table and various locations of its image in the image plane.

(b) An edge-follower program and a line-fitter program, which are also modules of the hand-eye system, are run on each picture stored and the location of the vertices of the object in the i.c.s. is determined.

(c) A guess of the values of all the parameters of the full model is made. Using these values and the pot readings associated with each picture, the transformation from t.c.s. to i.c.s. is computed. Using the transformation

and the locations of the vertices in the t.c.s. stored with each picture, a prediction of the vertices' locations in the i.c.s. is computed. The prediction is then compared to the i.c.s. locations obtained in step (b) and a root mean square, (RMS), distance error is computed, averaged over all vertices and all pictures.

(d) Step (c) is repeated while the values of the parameters are changed so as to make the RMS error smaller in each iteration. Two kinds of parameter minimization routines are used. Neither needs to explicitly compute the values of the first or second derivatives of the RMS error with respect to each parameter of the model.

The calibration process is done separately for each lens. The R.M.S error that is achieved when 20 pictures are used is 1 to 2 raster units.

The values of some of the parameters differ slightly when calibrated for different lenses, although they are known to be independent of lens type. For example: P1,KX,PPOTØ,TPOTD. This points to another advantage of calibrating the whole set of parameters together: there is enough "freedom" to compensate for changes in some parameters which depend on the lens type, but were not taken into account in the model.

Since the transformation equations do not use the lens focussing equation but only the magnification, the model used by Sobel has a scaling variable "fmy" which can be defined in the terms of our model as: fmy=foc*KY and a parameter MART defined by: MART=KX/KY. Also two other parameters, FPOTØ and FPOTD, are used instead of FOCLENØ and FOCLENG to convert the focus pot reading directly to FMy. Hence the model used by Sobel has 16 parameters: P1, P2, P3, DP1, DP2, DP3, PPOTØ, PPOTD, TPOTØ, TPOTD, FPOTØ, FPOTD, SWING, MART, PP1, PP2.

4.3 Calibration updating, general notes

After a calibrated model was obtained by Sobel, we used the corner-finder described in Chapter 3 to find the c.t.c. of some points on the table. They differed from the u.t.c. of the same points by about Ø,2 inch. These errors are 2 to 4 times larger than that corrsponding to an error of 1 raster unit on the image plane (the correspondence depends on the pan and tilt angles) quoted by Sobel. This result is resonable since the errors computed by Sobel were for precisely the points used in the calibration process.

It was noted that after some time (on the order of a week or two) the performance of the model deteriorates and the errors grow to 0.5 to 1.0 inch. Some of the parameters are not modelled correctly any more. One remedy is to repeat the whole calibration process every few days. This remedy is very time consuming (about 30 to 45 minutes of computing time) and very frustrating. An alternative is to find the parameters that have changed and update them only. The updating of some parameters could also partially compensate for changes in other parameters. Hopefully this process would be much shorter. Also, it could easily be automated since the existing calibration, although not good enough anymore, is sufficient to direct the pointing of the camera, focus it and search for the image of the vertices using the corner-finder described in Chapter 3 all under computer control with no manual intervention. This solution was pursued and is described below and in subsequent sections.

We can divide the parameters into three groups:

(a) Parameters which depend on the physical dimensions of the camera assembly: P1,P2,P3,DP1,DP2,DP3,

(b) Parameters which depend on the operation of electronic circuits: SWING,PP1,PP2,KX,KY.

(c) Parameters which depend on "electrical" properties such as the pots and the A/D converter power supplies, the pots' linearity, etc.: PPOT0, PPOTD, TPOT0, TPOTD, FOCLEN0, FOCLEN. The parameters FPOT0 and FPOTD used by Sobel depend on both the focus pot "electric" characteristics and the electronic scanning circuits.

The first group of parameters is the least susceptible to change with time. The other two are more susceptible to change. Of the second group parameters, SWING (which is about 1 deg.), PP1 and PP2 will have only second order effect on the accuracy of position predictions, even with a change of 20-30% in the case of PP1 and PP2 or 100% in the case of SWING.

KX, KY and the third group parameters have similar effects on the accuracy. The accuracy is more sensitive to the changes in the pan and tilt pot constants than to the focus pot constants or KX and KY as can be seen in the following example: A change in PPOT0 of 1% of its total range, which is at least 60 deg., will cause an error in position of 0.4 inch at a point on the table which is 40 inches from the lens center (a typical distance). A change of 1% in KX will cause a position error of 0.1 inch under the same conditions as above if the image of the point is at the border of the frame. This error will decrease linearly with the distance of the image of the point, from the point (PP1,PP2).

Because of the above considerations it was decided to start with the development of an updating scheme for the third group parameters, first the pan and tilt pot constants and then the focus pot constants.

4.4 Pan/tilt head calibration updating

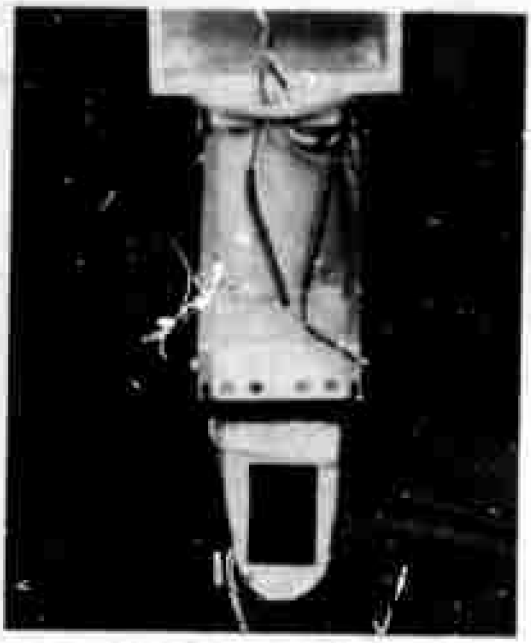
We have two routines to update the calibration of the pan and tilt pots. One of them uses the corners of cubes placed on the table at known locations. This routine is not as completely automatic as we wish it to be since the cubes are manually placed at those locations. It aims at reducing the differences between the coordinates of a point in the t.c.s. as approximated by the user and the approximation given by the c.t.c.

The other routine uses the arm itself instead of the cubes. Since, in general, the image of the hand at the end of the arm is still too complicated for the vision programs to analyze, a black rectangle was painted on each of the fingers (the hand-mark). Also, the hand is rotated so that one finger is facing the camera, hiding the other finger and parts of the hand, (see Figure 28(a) and compare with Figure 28(b)). If the calibration has not changed too much, we can ask the corner-finder of Chapter 3 to search for the corner of the hand-mark without its getting confused by other features of the arm. Demanding a match of form and relative intensity helps the corner-finder to recognize the hand-mark. Since there are no other objects on the table when the calibration updating programs are run, we can let the corner-finder search in a fairly large area and so handle large parameter changes. We are limited however in one direction by the presence of the upper part of the hand.

By using the arm we achieve two goals: automating the process and reducing the coordination errors. In effect, we Assume here that the deterioration of the pan and tilt parameters is the only cause of the coordination errors. (We make a similar assumption for the focus pot in the next section). We update the parameters to minimize the measured coordination errors. (Actually we do not measure them but only their effect on the location of the hand-mark corner in the i.c.s. (see step (c) below and Appendix A).

The updating is done in the following steps:

(a) The camera is centered and focussed on the next position of the arm (or the next cube). Centering is performed to ascertain that the image of the corner sought is in the frame and that the effect of the changes in internal model parameters is minimized. The precision of the centering process does not affect the updating.



(a) HAND FACING THE CAMERA

(b) HAND IN GENERAL POSITION

Fig. 28. THE HAND AS SEEN BY THE CAMERA.

(b) The camera transformation is computed for the new (centered) position. The coordinates of the image vertex and the form of the image corner are then computed using this transformation.

(c) The corner-finder is called and given the above information. When using the cubes, we chose dark colored cubes on a light background or vice-versa. When using the hand the corner of the hand-mark is sought. In both cases a relative intensity match is demanded in addition to form match as mentioned before. If the corner is not found, as happens occasionally when the calibration has not been updated for some time, the routine goes back to step (a) and to the next cube or arm position.

After the corner is found the error between its actual and predicted location in the i.c.s. is computed. This error is used in two ways: First, we assume that at least part of the error is systematic and once we add the error to the predicted location of the next corner before calling the corner-finder. This turned out to be a real winner. Second, we compute what the pan and tilt angle should be in order that the actual and predicted locations will coincide. We record these angles with the actual reading of the pan and tilt pots.

(d) After data has been collected from all the cubes or arm positions (less the cases where the corner has not been found), two straight lines are fitted to two sets of points with the pan and tilt angles as abscissas and the pan and tilt pot readings as ordinates. The coefficients of the two lines are the updated pots' parameters for pan and tilt.

4.5 Focus calibration updating

As noted before, Sobel's model combined KX and KY with foc because there was no way to distinguish between them. One reason is that when the measurements were taken for the calibration, the camera was focussed on the calibration object as a whole and not on any particular point which could be used to calculate separately the coefficients for the focus pot. On the other hand, we need exactly those separated coefficients in order to calculate the focus pot setting needed to focus the camera at a particular range.

In this case also, either a set of cubes placed on the table, or the arm is used. The positions are chosen to present a variety of ranges. The updating is done in the following steps:

(a) The camera is centered on the next position of the arm (or the next cube),

(b) The camera is focussed on the corner of the hand-mark on the finger (which is made to directly face the camera), or on the next cube, by moving the vidicon (i.e changing foc). In the beginning the focussing was done manually. Then we added an automatic focussing routine which was developed by J.M. Tenenbaum [28]. (See Appendix B). In the latter case the corner-finder is called first to exactly locate the corner before the automatic focussing routine is called. The focus pot reading is then recorded,

(c) The range from the lens center to the above corner is computed, and then using the focus equation the corresponding value of foc is computed and recorded,

(d) After all data has been collected a straight line is fitted to the set of points with the computed values of foc as abscissa and the focus pot's readings as ordinate. The coefficients of this line are the updated focus pot parameters,

It was noted before that when calibrating for different lenses the parameters were not forced to take the same values even if there was no reason for them to be different for different lenses. In the case of the focus pot parameters we have to stray from this principle. The reason is that for the small focal length lens, 1 inch, the depth of field is large enough so that we can not fit a line to the readings with enough confidence. Hence we use the fact that the slope of the line has indeed the same value for all lenses and only the bias changes. Thus a long focal length lens (3 inch) is used to calibrate the slope. For the smaller focal length lenses only the bias is updated by the above process using the slope already obtained,

Since we do not yet have a good external way to measure the quality of focussing , we have to rely either on the self scoring automatic focussing module or on a human observer and the monitor to judge that the updating process is adequate,

4.6 More remarks about updating

When the degradation of the parameters is so large, that the corner-finder cannot find the hand-mark, either because it is out of focus (if the degradation is of the focus pot parameters) or because it is out of the search space (if the degradation is of the pan and tilt pot parameters), or both. In this case we use the manual mode of the update programs,

The manual mode for the focus pot updating program was described above. In the pan and tilt pots update program we have a provision for manually positioning the window on the desired corner. Usually this has to be done only for the first position of the arm because of the "instant adaptation" described in step (c) of Section 4.4.

If the hand mark is out of focus, the focus updating routine (in manual mode) has to be called first. After that the pan/tilt updating routine can be called. Then we should call the focus updating program once again to make the model more accurate (this time it can be called in automatic mode). The above cycle can be repeated two or three times until no more improvments can be made.

There are some alternatives to the use of routines which are executed specifically for the purpose of updating:

(a) The coordination errors (i.e differences between the c.t.c. and h.t.c.) which are detected during regular operation of the hand-eye system are stored indexed by the place over the table where they occurred. Then they are used subsequently for this region of the table. Interpolation between locations for which the coordination errors were stored should be considered. A similar scheme is used in the visual feedback tasks to augment the updating done with the routines described in this chapter.

(b) The values of the errors are used to update the model each time an error above some threshold is detected. The correction term should be weighed appropriately, otherwise the model will "thrash".

(c) The values of the errors are stored regularly. If a degradation of the model is detected (based on more than one error), then all the stored errors are used to update the model.

For the pan/tilt updating, the equations for (b) and (c) are more complex than those described in Appendix A for the method of Section 4.4, since the errors are not necessarily observed near the piercing point of the principal ray.

The advantage of the alternative methods is that no special programs, or the time to run these programs is needed. The disadvantage is that we have to make sure that the coordination errors were detected and measured with high reliability. This is not a problem with the special purpose programs since the environment is very simple, but it could cause trouble during regular operation, where the environment is often cluttered.

CHAPTER 5: VISUAL FEEDBACK TASKS

5.1 Introduction

5.1.1 General notes

The purpose of the visual feedback (v.f.b.) tasks is to increase the preoision of the manipulations done in the hand-eye system. The feedback currently does not take into account the dynamic aspects of the manipulation (see futher discussion in Section 5.1.2).

The tasks are carried out in the context of the general tasks that the hand-eye system can currently perform, i..e. the reoognition and manipulation of simple planar bounded objects, whose faces are uniformly painted (without texture) with various colors, on a background also of uniform shade (usually dark). The lighting is supplied either by the regular fluorescent overhead lighting which is somewhat but not completely diffused, or by three Quartz Iodine lamps which can boost the usable intensity dynamic range but also create harsher shadows which are harder to cope with.

The manipulations that we sought to make more precise with the addition of visual feedback are grasping placing on the table and stacking. From now on when we mention these three operations, it will be meant that they are done under v.f.b. control.

The precision obtained is better that .1 inch. This value should be judged by comparing it with the limitations of the system. The resolution of the imaging system with the 2 inch lens is 1 mrad, which, at an operating range of 30 inch corresponds to .03 inch. The resolution of the arm position reading (lower bit of the A/D converter reading the first arm joint potentiometer) is also .03 inch. But the noise in the arm position reading corresponds to .05 inch. When we tried to achieve preoision of .05 inch, the feedback loop was executed a number of times until the errors randomly happened to be below the threshold.

The rest of this section is devoted to the discussion of some general problems pertaining to all three tasks. The next three sections describe in detail the three tasks as implemented for cubes. Since the termination of the research described here, the capability of the program has bee extended by R. Davis to handle other objects (wedges and slabs) and one more task of inserting objects into holes of the same form.

5.1.2 Dynamic feedback

The question of whether the visual feedback, or in general the response, is dynamic or not, is sometimes more semantic than real. What the person asking the question means in this case is, does it seem to be continuous? The question then is really that of cycle time or sampling period. A cycle time of 20 msec will suffice to fool the human eye so that the response will be called "dynamic". Since the computations needed and the computing power that we now have cause the length of the cycle time to be several seconds, no attempt was made to speed it up by programming tricks, use of machine language, etc. With this cycle time the movement of the arm actually stops before we analyze the situation again, so that we do not have to take into account the dynamic aspect of the error.

In addition to computing power, the vidicon camera also presents some limitations to faster response. The short time memory of the vidicon which helps us (persons) to view the TV monitor, will "smear" a fast moving object. If the scene is bright enough a shutter can be used. If the vision can be made sufficiently fast and accurate, the control program [18] currently used to run the arm dynamically could be expanded to incorporate visual information.

5.1.3 Hand in view

As already mentioned, one of the characteristics that distinguishes our visual feedback scheme is that the analysis of the scene, to detect the errors to be corrected, is done with the hand (which is still holding the object) in the scene. In the grasping task the presence of the hand is inevitable. In other tasks, for example stacking, being able to analyze the scene with the object still grasped helps to correct the positional errors before they become catastrophic (e.g the stack falls down). Also some time is saved since there is no need to release the object, move the arm away, bring it back and grasp the object again. We pay for this flexibility with increased complexity of the scene analysis.

The difficulty is lessened by the fact that the hand-mark (which is used to identify the hand) has known form and relative intensity which helps to locate it in the image. In the grasping task the ability to recognize the hand is necessary. The task is essentially to position the hand at fixed location and orientation relative to the object to be grasped.

Also, we have found it to our benefit to locate the hand first in the other tasks also. After the hand-mark is found, we use its location to predict more accurately the locations in the image of the edges of the object held by the hand. We can do this since the object was grasped with visual feedback, and therefore its position and orientation in the h.c.s. are known. (We also take into account the remaining errors of the grasping task). We use the coordinates in the h.c.s. to find the h.t.c. of the edges. From the h.t.c. we predict the image coordinates. We correct the prediction by the difference between the predicted and actual location of the hand-mark found in the image. This is the right correction since the prediction of the hand-mark location in the image was obtained by exactly the same calculations. (This is very similar to the "instant adaptation" described in Section 4.4).

Moreover, after the hand-mark has been found, it will not be confused with other edges in the scene.

5.1.4 Inference of three dimensional information

In the pan and tilt calibration updating programs we assumed for convenience that the coordination errors were caused solely by deterioration, over time, of the pan and tilt pot constants. By executing the calibration updating program we removed part of the coordination errors (mainly the part that had systematic characteristics over various locations in the t.c.s.).

In the v.f.b tasks we assume, again for convenience, that the remaining coordination errors (which will be called simply, errors) are caused by the arm.

These two assumptions are made for covenience but really they are equivalent.

As explained in Section 4.1.2, the c.t.c. approximation is used for reference. Since we are using only one camera (one view), we cannot measure directly even differences of locations of two neighbouring points. Hence the three-dimensional (3-D) information has to be inferred from the two-dimensional (2-D) information available in the image and some external information.

One assumption that can be made concerning points on the hand or on an object held by the hand, is that the positional error of the arm in a particular direction, for example height above the table, is much smaller than the errors in the other directions. This assumption can then be used to infer the errors in the other directions from the 2-D image. If, however, such an assumption is made

incorrectly, the error in the direction which we assumed errorless will cause errors of the same order of magnitude to be inferred for the other directions. We do not have any reasons to justify this assumption in our system and other external information has to be used.

We assume, however, that the main errors are in position and not in orientation. A small error in orientation when multiplied by the length of the opening between the fingers will generate a position error of the hand-mark which can be neglected relative to the position errors of the arm. For similar reasons we also assume that the error in measuring the distance between the fingers can also be neglected.

The external information, which is used in the v.f.b tasks, is supplied either by the touch sensors on the fingers or by the fact that an object is resting on the table-top or on another object of known dimensions (this is the famous support hypotehsis mentioned in Section 4.1.2). The touch sensors help us to determine the plane containing the hand mark from the known position of the touched object. The support hypothesis gives us the plane containing the bottom edges of the top object. More implementation details are given with the descriptions of the various v.f.b tasks.

5.1.5 Hand-Eye coordination.

To facilitate the scene analysis we use here one of the methods for calibration updating mentioned already in Section 4.6. We store the coordination errors found during regular operation of the system (specifically, carrying out v.f.b tasks) and use them subsequently as described below. We justify this by the assumption that no major change of the arm and camera hardware is going to occur during the short time of operation. If needed we can add the time that the errors were stored and find and store them again after they become too old. We do not currently attempt to use the stored coordination errors for actual updating of the camera or arm models.

Actually we do not store the coordination errors themselves but two quantities closely related:

(a) After an object is grasped and then released, the h.t.c. of the reference point of the hand (midpoint between the touch sensors) is stored with the object so that the next time we want to grasp the object (or stack on it), we use the h.t.c. rather than the c.t.c. to position the arm. However, if the object was never grasped before, the c.t.c. are used. The v.f.b is used in both cases (and not only the second) to check and correct the grasping. The task

is simpler if the h,t,c, are already known, because the repetition errors of the arm servo are smaller than the coordination errors.

This last characteristic also explains the following fact: The initial error in stacking is smaller if the supporting object has already been grasped once by the hand and hence its h,t,c, already stored. In this case the main part of the coordination error is already found and corrected by the v,f,b during grasping. On the other hand, if the supporting object was never grasped before, the coordination error has to be corrected by the v,f,b control during the stacking which is more complex and difficult than the control of grAsping. If really needed, the object to be stacked upon can actually be grasped first, (with v,f,b), to find its h,t,c,, and then released to continue with the stacking task.

(b) Before an object is to be grasped or stacked upon, the arm is positioned above the object and the hand-mark is sought as was done in the pan and tilt calibration updating. The hand is poisitioned high enough above the object so that the corner-finder does not confuse the hand-mark with the object. After the hand-mark is found, the differenoe between the ooordinates in the i,c,s, (image coordinate system) of the predicted location of the hand-mark and the location where it was aotually found is stored. This is the same quantity stored in the pan and tilt calibration updating program for the "instant adaptation". The same is done for the place on the table where an object is going to be placed.

The table is divided into 4 inch squares, (there are 100 squares), and the corrections are stored with the souare over which they were found. When we Subsequently look for the hand-mark over this part of the table, the stored differences are used to correct the prediction. Since we now have a corrected prediction, the dimension of the window, or the search space used. can be made smaller .

Each time that the hand-mark is found again, the differences between predicted (before correction) and actual locations in the image are also used to update the stored corrections.

Note that we cannot infer the three dimensional coordination error from the two dimensional difference, as measured above, since there is no external information that can be used in this case.

5.1.6 "State of the world" updating

The state of our simple world (table, arm, camera and objects) is stored for use by the various modules. The state of the arm and camera are stored automatically by the arm and camera control programs every time a change is attempted regardless of the degree of success. The driver module or other modules which manipulate the objects have to supply and update the state of the objects during and after each manipulation.

The components of the state of the world that have to be updated during the execution of the v.f.b tasks are described in the following table without details of implementation. Currently we use a simplified structure, suited to cubes only, but eventually the general world model of the system, stored in the associative structure supplied by SAIL, will be used.

(WM1) A pointer to the object currently held by the hand.

For each 4 inch square part of the table face:

(WM2) The l.c.s. corrections, described in the last section, for the square.
(WM3) An indication of whether the l.c.s. corrections were already obtained for this square.

For the object currently held by the hand:

(WM4) The position and orientation of the object in the hand coordinate system. (For cubes, the remaining errors of the grasping task in the two directions parallel to the edges of the hand-mark).

For each object:

(WM5) Description of the prototype of the object. (For cubes, the size of the cube).
(WM6) The position and orientation of the object in c.t.c. (For cubes, the position only, orientation is inferred from WM14).
(WM7) An indication of whether WM6 was updated for the current location of the object.
(WM8) The position and orientation in h.t.c. of the object if known. If not, WM6 is copied as a first approximation. (For cubes, only the position, since we assumed negligible error in orientation).
(WM9) Pointers to objects directly supported by the object (currently we allow only one).
(WM10) Pointers to the support of the object which might be other objects (again we currently allow only one), the table or an indication that the object is not supported.

(WM11) An indication of whether the object is stable and hence can be released.
(WM12) An indication of whether the object is currently held by the hand.

For each face of all objects:

(WM13) An indication of whether the face is visible to the camera. (For cubes, the sine of the angle between the line of sight of the camera when centered on the cube and the normal to the "most visible" vertical face, described in the next section. The visibility of other faces or edges are computed from this information).
(WM14) The normal (in o.t.c.) to the face. (For cubes, the normals to the two vertical visible faces).

Some of the components are redundant. For example, the normals to the faces can be computed from the prototype information, (in WM5), and orientation (in WM6). This is an example to the general problem of "store vs recompute": when to store enough redundant information to speed execution, when to store information only after it was needed once, for possible subsequent use, or when to compute it again each time it is needed.

We now proceed to discuss the problems of updating the state of the world. If each operation was totally reliable we could update the state of the world after planning the change, or after carrying out the operation but before the information is needed by other modules, or at any time in between. If all the failures were of a semi-permanent nature, (something has to be externally fixed before the modules could carry on), then again we could update any time before the information is needed and quit altogether at the first failure.

Overall, the reliability of the manipulation is very good. However, from time to time the arm (or hand) will fail because, for example, of rare circumstances of the time sharing system, or transients in the hardware. Most of these failures do not have catastrophic effects and the arm simply stops moving and indicates the source of failure. On some occasions, blindly repeating the operation is the wrong thing to do, for example when executing an incrementally specified movement of the arm. In other cases we would like to change or abort the manipulation after the failure. Therefore we need to time the updating carefully so that the updated information would be available immediately after failure but not prematurely.

For the same reasons we need both WM9 and WM10, or both WM12 and WM1, which are used to give correct information for different modes of failures, by not being updated at the same time.

For example: We use WM1 to indicate that the object pointed to may be in the hand and therefore we need to release it before moving the arm but it is not guaranteed that the object is firmly grasped and hence we have to grasp it if we want to move it with the arm. On the other hand WM12 indicates that the object is firmly grasped by the hand and hence will move with it.

If we tried to grasp the object and failed we would update WM1 because the grip might be firm enough to move the object a little before it slipped from the hand. WM12 would be updated only after the grasping is successfully completed. Similarly, if we tried to release the object and failed, WM12 would be updated but not WM1 which will be updated only after the release was successfully completed.

The various components of the state of the world are updated at the right times so that the following rules can be observed and be truly satisfied. Some of the reasons for these rules are obvious, others will become clear in the description of the various tasks.

(I) The c.t.c. of an object have to be known in order to grasp it or stack on it.

(II) An object is not released unless it is known to be stable which means that it is resting on the table or supported by other object(s) in a stable position. In the latter case it necessarily means that its c.t.c. are known.

(III) An object is grasped, placed on the table or stacked on another object only if the appropriate i.c.s. corrections are known.

(IV) An object that is supporting another object cannot be grasped or stacked upon. In more complex situations when more that one object will support or be supported by other objects, we will have to check more complex conditions: can object A support in a stable position an additional object B? Will the removal of object A cause any object to become unstable?

(V) The vicinity of the location on the table where an object is to be placed has to be empty of other objects. A v.f.b task of abutting objects has not been implemented yet.

Some of the rules and update timing are simple because we currently have only one arm with one hand that can hold and manipulate only one object at a time. For example: When we have two arms we will allow one of them to hold an object to make it firmer even if it supports another object, but not to move it, etc.

The appropriate rules are checked before each operation is attempted. If rule (III) is not satisfied the program will try to find the corrections as described in the last section. Failure to satisfy the rules causes failure of the v,f,b tasks unless otherwise stated in the tasks' descriptions.

For brevity's sake we will not mention these facts again in the description of the tasks. We will indicate the points in execution where various components of the world model are being updated.

5.1.7 Manipulations not under visual feedback control

The manipulations described in this section are not carried out under direct v,f,b control. They are used by some or all of the v,f,b tasks, and described here in order not to repeat the description each time they are used. We will indicate their usage by writing their names in capital letters.

Some movement of the arm or the hand will at least be attempted in each of the following tasks. Hence, one component of the world model will be changed regardless of the degree of success. We will mention this change before we mention the movement or the command to the arm controller.

UNGRASP

If there is a cube pointed to by WM1, it is checked for stability. If the cube is unstable, failure is indicated and the task is terminated. Otherwise we indicate in WM12 of the cube that was grasped, if any, that it is no longer firmly gripped. Then we send the arm controller the open-hand command. If it succeeds we update WM8 for the cube that was grasped (if any), and delete the pointer from WM1. The required distance between the touch sensors is given to the procedure as a parameter.

LIFT

In this manipulation the cube is lifted about .5 inch above its support. First, we indicate for the cube in the hand that it is not supported any more (in WM10), that it is unstable (in WM11) and that its c.t.o. are not updated (in WM7). Then we send the arm controller the command to lift the cube. If it succeeds we delete the pointer from

WM9 of the supporting cube (if any).

PUTDOWN

This manipulation places a cube on its support using force feedback of the arm to stop pushing down. First, A pointer to the cube held in the hand is placed in WM9 of the cube placed upon (if any). Then we send the arm controller the command to place the cube. If it succeeds, a pointer to the support is placed in WM10 of the cube in the hand. If the cube was placed on the table it will also be marked stable (in WM11).

This manipulation actually supplies the justification for the suppport assumption when analyzing scenes with an object still held by the hand. Another way to look at this manipulation is to note that it actually eliminate the coordination error in the z-direction of the t.c.s.

MOVETO

This manipulation moves the arm with the cube grasped in the hand (if any) to a location and orientation given to it as parameters. First the cube in the hand is marked unstable, it is indicated that its c.t.c. are unknown and the pointer to its support is removed. Then we send the arm controller the command to move. If it succeeds the pointer in WM9 of the cube (if any) that supported the cube in the hand (if any) is removed. Each trajectory starts and ends with the fingers pointing down and moving in a vertical direction.

To complete the list we will describe the routine used to find the hand-mark.

FIND_HAND

In this routine we are looking for both corners and not for only one as in the calibration updating program, since now the scene is complicated by the presence of other objects.

The camera is centered on the hand-mark. Using the camera and arm models, the locations of the images of the two lower corners of the hand-mark in the i.c.s. are predicted. The predictions are corrected by the stored i.c.s. corrections (WM2) if they have already been obtained. Also the form of the image of the corners is computed as was done in Section 4.4. The predicted width of the hand-mark in the image is stored.

Using the information computed above, the right side corner is sought first, using the corner-finder of Chapter 3. If the corner is found, the error between its predicted and actual locations is used to update the prediction of the

location of the left side corner which is now sought. If the right corner is not found we look for the left one first (see block diagram in Figure 29).

This algorithm is an example of the use of information about a relation between two features to be found in addition to information pertaining to each feature alone.

We check that we found the corners belonging to the hand-mark, and not those belonging to a cube which might have very similar form, by comparing the distance between the corners in the image with the stored predicted width.

The last computed prediction errors are stored in WM2, with indication in WM3 if needed, for the appropriate square part of the table.

In the following description of the v,f,b tasks, if any of the above manipulations or other operations, like centering the camera, finding a line, etc., fails, it will cause failure of the v,f,b task unless otherwise stated.

5.2 Grasping task (GRASP)

The task is to precisely grasp a cube of known (c,t,c,) position and orientation in order to move it and place it somewhere else, or stack it on another cube. The precision is needed in order not to drop the cube in mid-trajectory (which can happen if the cube is grasped too close to an edge), and in order that its position relative to the hand will be known. This information is used in the other two v,f,b tasks. We try to grasp the cube on the mid-line between the faces perpendicular to the fingers, half way above the center of gravity.

Note that in one direction (namely perpendicular to the fingers) the hand does its own error correcting! When the hand is positioned over the cube with maximum opening between the fingers (2.4 inches between the tips of the touch sensors) and then closed, the cube will be moved and always end in the same position relative to the hand, independent of the initial position. (See Figure 30). This motion is sometimes disturbing (e.g when grasping the top cube of a stack, the movement can cause it to become unstable before it is fully gripped and it will fall off the stack), and hence no use is made of this feature. Instead we also correct errors in this direction so that when a cube is grasped, it is being moved from its original location by less than the tolerance of the feedback loop.

The grasping is done in the following steps (The numbers in

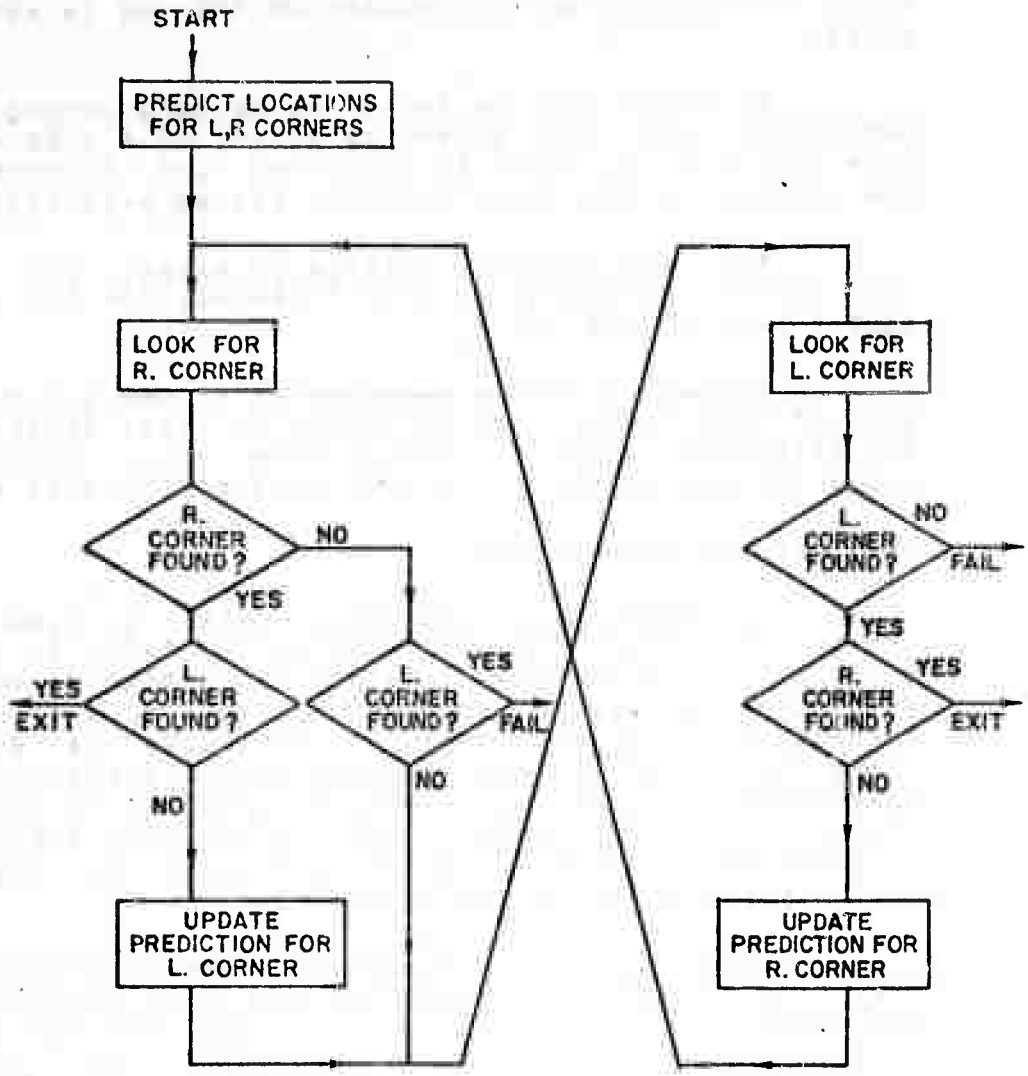


Fig. 29.

BLOCK-DIAGRAM OF HAND-MARK LOCATION

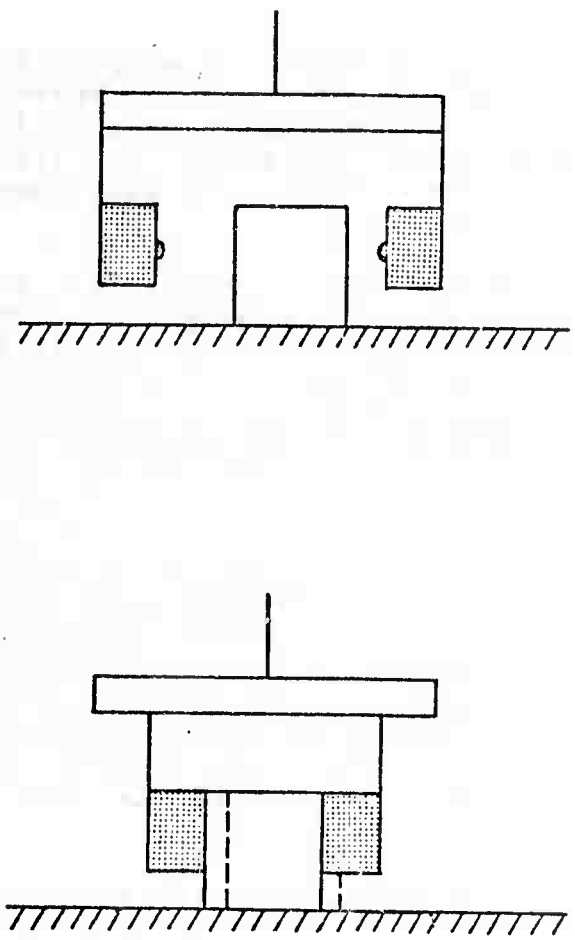


Fig. 30.

SWEEPING MOTION OF THE FINGERS

each step refer to figures):

(a) The fingers are fully opened (by UNGRASPing) and and the hand is MOVEdTO over the center of the cube so that the fingers are parallel to the cube's sides. The cube has at most two vertical faces visible to the camera. One of them is more visible to the camera in the sense that its normal is pointing closer to the direction of the camera center than the normal of the other one. We call this face MVVF, (most visible vertical face), the other is called SVVF (second visible vertical face). The hand is oriented so that the fingers are parallel to the MVVF. This will help to locate the hand-mark in the image more precisely. (31(a)) and 32(III)).

(b) The touch sensors are enabled and the hand is closed slowly (at about 1/4 of the usual speed or about 1 inch/sec) till one of the fingers touches the face of the cube. As mentioned before, the touch is light enough so that the cube is not moved. The touch sensors are then disabled. (31(b) and 32(IV)).

(c) Using the following information (for which we assume that errors can be neglected): The fingers are parallel to the cube's face; The distance between the tips of the sensors after the closing motion of the fingers is stopped; The thickness of the fingers between the tip of the sensor and the hand-mark, the equation for the plane containing the hand-mark facing the camera is computed in the t.c.s.

(d) The hand-mark is sought using FIND_HAND. After the two corners of the hand-mark have been found, the camera transformation is used to compute the corresponding rays in c.t.c. These rays are intersected with the plane found in step (c) to give the c.t.c. of the corners. To verify that the corners found do belong to the hand-mark, we check that they have approximately the same height, and that the distance between them corresponds to the width of the hand-mark.

(e) Using the information already used in step (c), the position errors of the hand are computed. If the magnitude of the errors in all three directions are less than a threshold (currently .1 inch), the task is finished, we go to step (f) and then exit. If the errors are larger, the hand is opened (32(V)) and the errors are corrected by changing the position of the arm appropriately. (31(e) and 32(VI)). We then go back to step (b) to check errors again. (32(VII)).

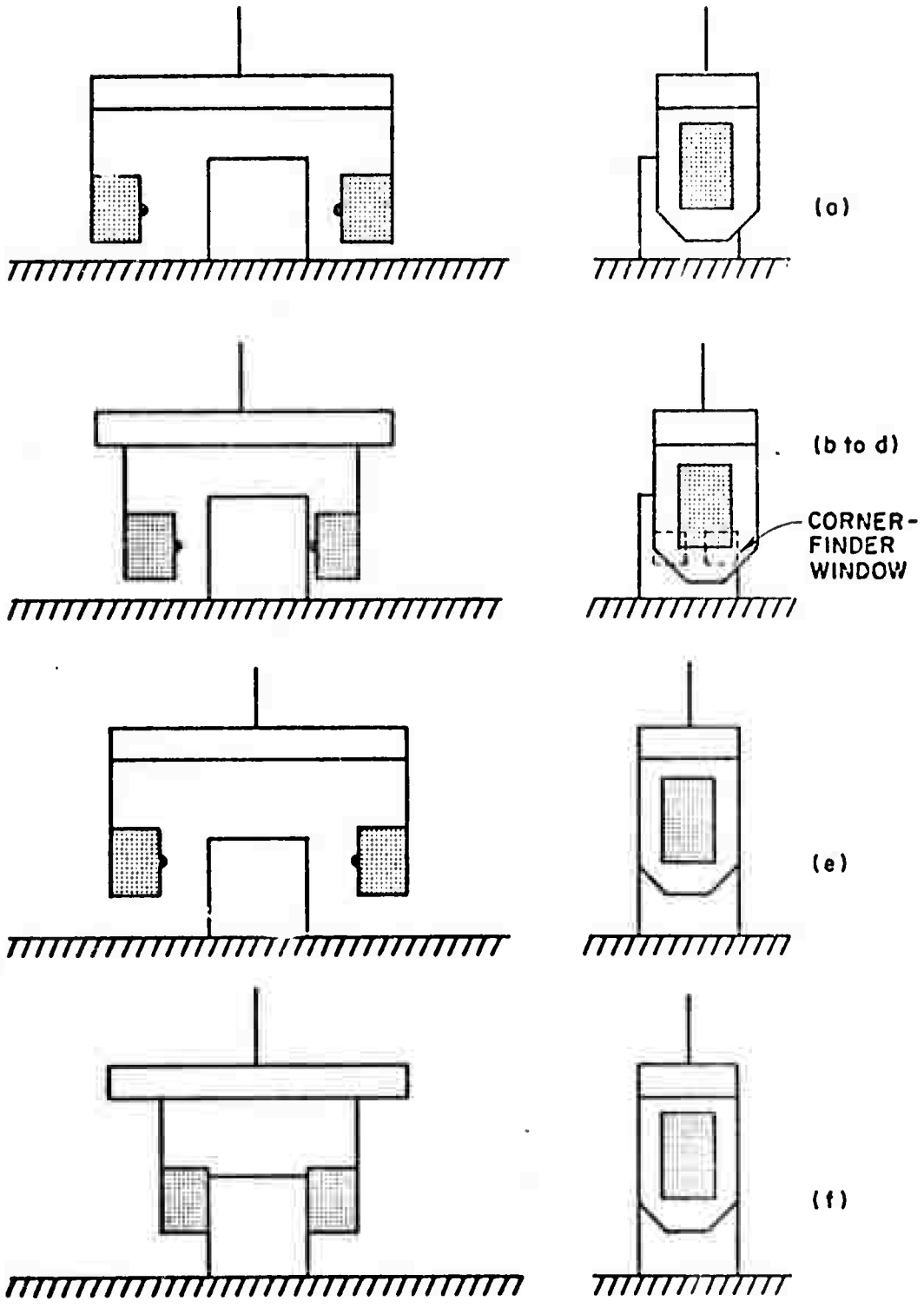


Fig. 31.

GRASPING TASK

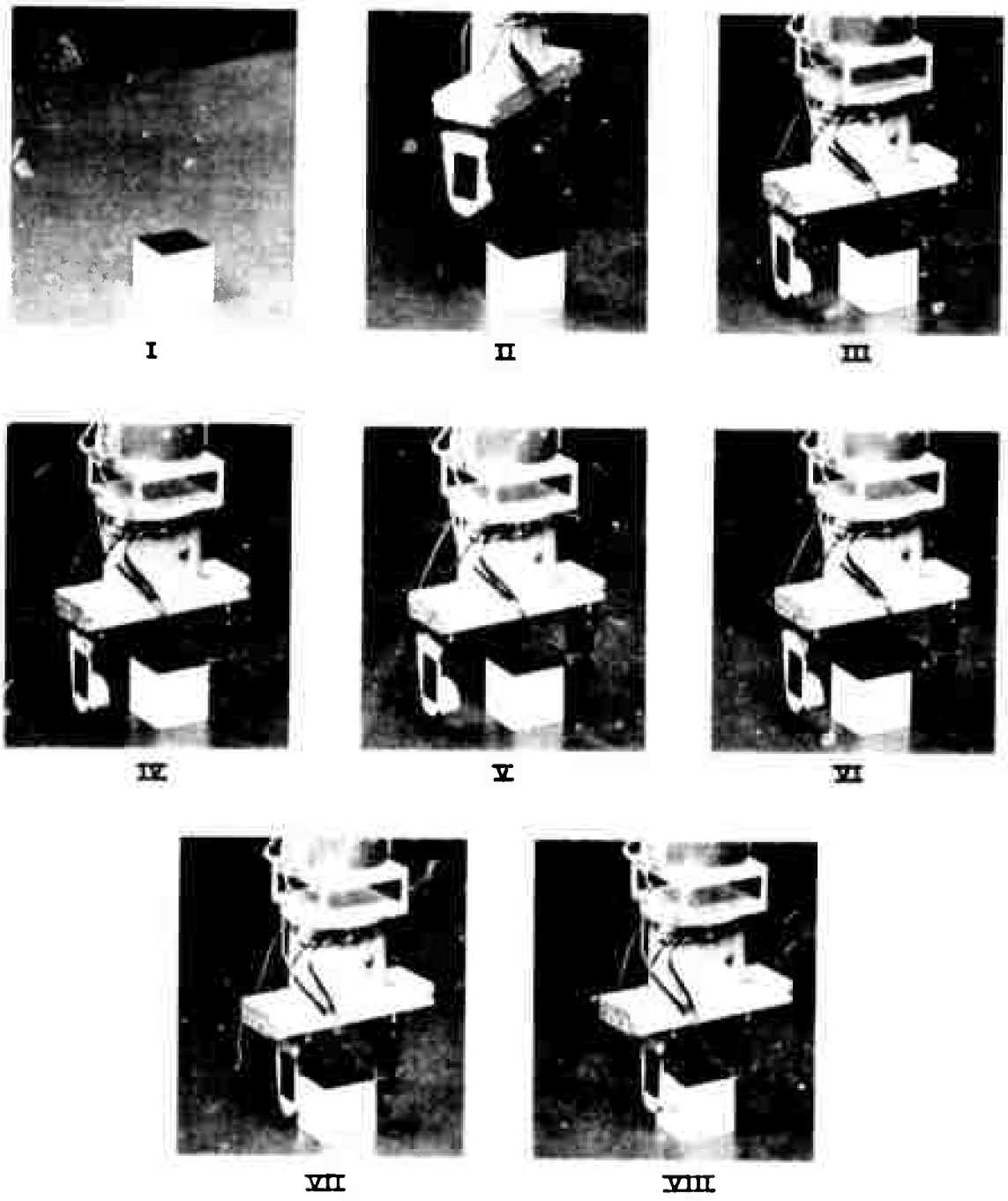


Fig. 32. GRASPING TASK.

(f) The pointer in WM1 is made to point to the cube. Then the arm controller is given the close-hand command. If this operation succeeds or fails the pointer in WM1 is made to point to the cube. If it succeeds (31(f) and 32(VIII)), the fact that the cube is now firmly gripped is indicated in WM12. The components of the remaining errors in the directions parallel to the edges of the hand-mark are stored in WM4.

5.3 Placing task

The task is to place the cube precisely at a given location on the table. With very minor modifications, it could be used to place the cube on any relatively large horizontal surface of known height which does not have any reference marks near the location where the cube is to be placed. In this case the support hypothesis is the only external information used. The task is carried out in the following steps:

(a) The cube is GRASPed (33(III)) and MOVEDTO a position above the table where the cube is to be placed. (33(IV)).
(b) The cube is PUTDOWN. (33(V)).
(c) The hand mark is located in the image using FIND_HAND.
(d) The camera is centered on the visible bottom edges of the cube.
(e) The locations of mid-points and the orientation of the images of the two visible bottom edges of the cube are computed using the hand trasformation, the information in WM4 and the size of the cube. The predicted location is then corrected by the amounts computed in step (c).
(f) The corner-finder is then used to locate the two lines in the image. The two lines are intersected to find the corner location in the image. Using the support hypothesis, the c.t.c. location of the corner is computed and compared with the required location. If the magnitudes of the errors are less than a threshold (.1 inch) in both directions then the task is completed. Otherwise the cube is LIFTed (33(VI)) and the error corrected by changing the position of the arm appropriately. (33(VII)). Then we go back to step (b) to check the errors again. (33(VIII)).

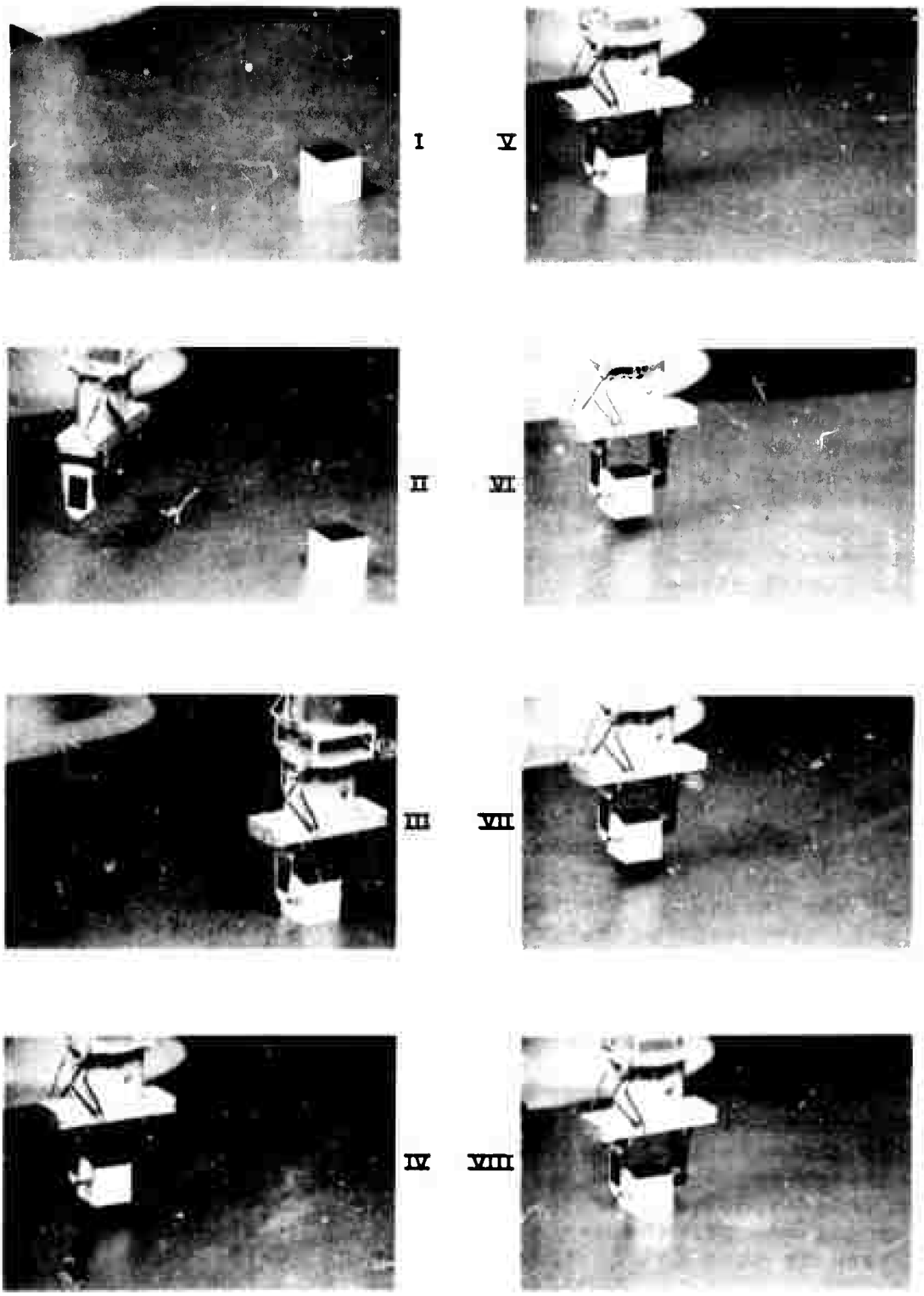


Fig. 33. PLACING TASK.

5.4 Stacking task

The task is to stack one cube on top of another cube so that the edges of the bottom face of the top cube (BFTC) will be parallel to the edges of the top face of the bottom cube (TFBC), at offsets specified to the program by the user or the calling module. The distances are specified perpendicular to the MVVF and SVVF.

The task is carried out in the following steps:

(a) The top cube is GRASPed. (35(III)).

(b) The camera is centerd on the TFBC. The mid-points and orientations of the images of the two edges of the TFBC, belonging also to the MVVF and SVVF, are computed. The corner-finder is used to locate these two lines in the image. The locations and orientations found are then stored. The two lines are intersected and using the known height of the cube, the c.t.c. of the location of the corner are found. Using the given offsets, the c.t.c. of the required positions of the corner of the BFTC and the center of the top cube are calculated. (34(b)).

(c) The top cube is MOVEdTO a location just above the bottom cube, oriented so that the hand-mark is parallel to MVVF of the bottom cube. The h.t.c. of the bottom cube (WM8) are taken into account if known. (35(IV)).

(d) The top cube is PUTDOWN on the bottom cube. (34(c) to (e) and 35(V)).

(e) The hand-mark is located in the image and then the two edges of the BFTC as done in steps (c) to (f) of the placing task. In this case, however, the edges of the TFBC will also appear in view. A simple algorithm is used with the information computed in step (b) to decide which of the lines are the edges of the BFTC. This simple algorithm can be deceived sometimes by the presence of shadows and "doubling" of edges in the image. We could make the algorithm more immune by locating the vertical edges of cubes also if they could be found.

(f) The two edges of the BFTC found in the last step are intersected to find the corner location in the image. Using the support hypothesis, the c.t.c. of the location of the corner is computed and compared with the required location computed in step (b). if the magnitudes of the errors are less than a threshold (.1 inch) in both directions then the task is completed. Otherwise the cube is LIFTed (35(VI)) and the error corrected by changing the position of the arm appropriately. (34(f) and 35(VII)). Then we go back to step (d) to check the errors again. (34(d) again and 35(VIII)).

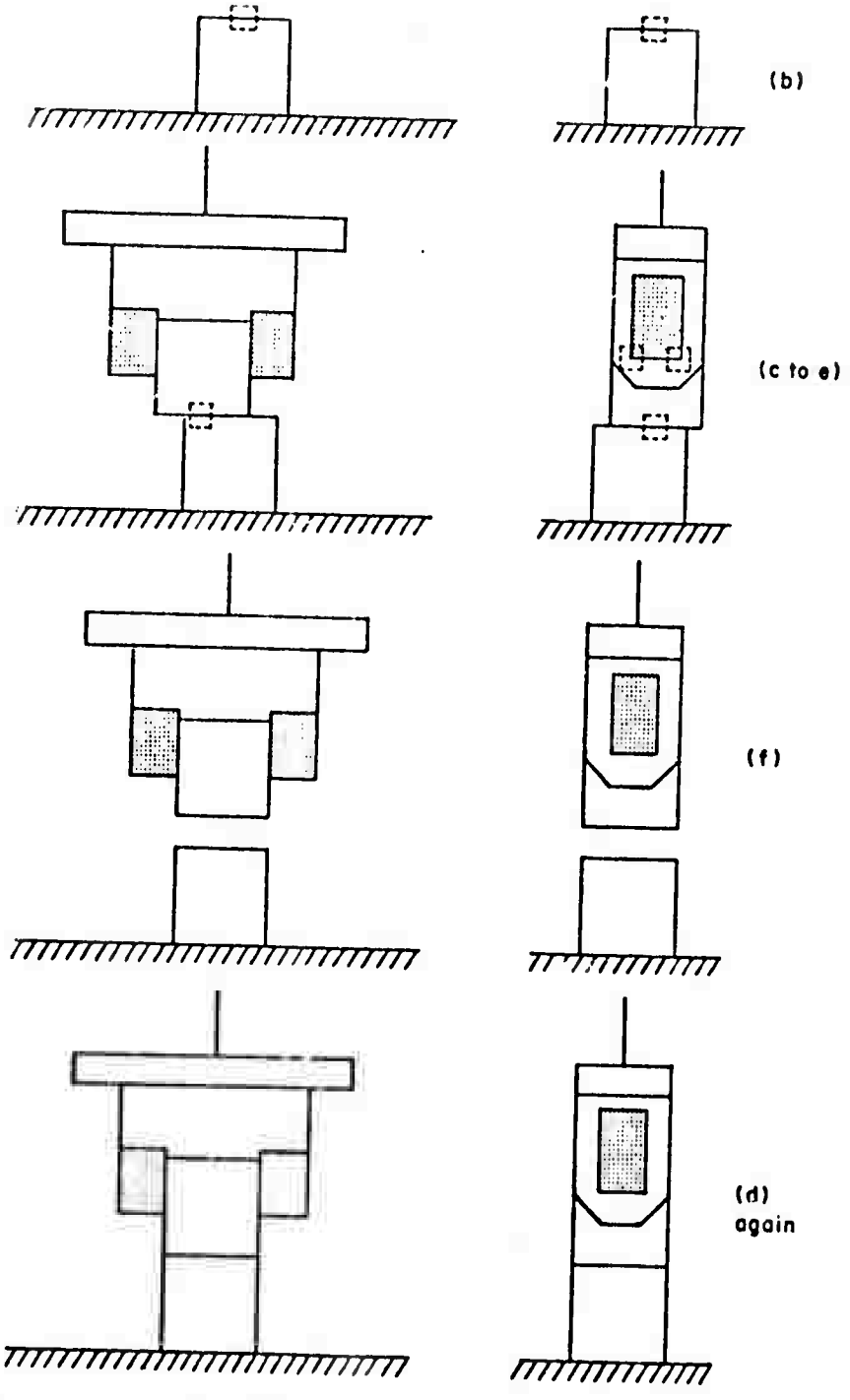


Fig. 2.

STACKING TASK

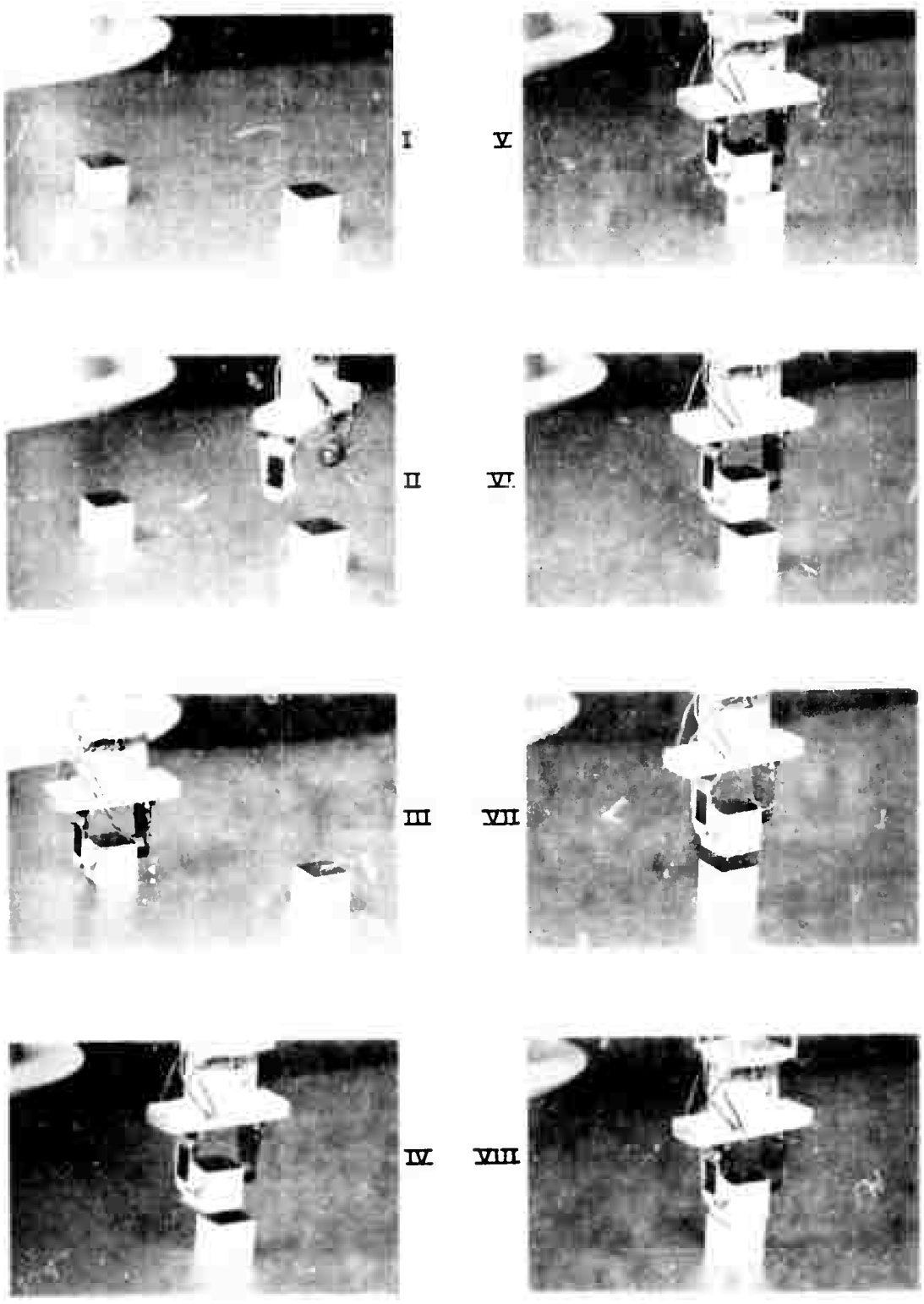


Fig. 35. STACKING TASK.

Instead of a bottom cube, we can specify to the program a square hole in a bottom object in which the top cube is to be inserted. In this case, when the top cube is placed on the bottom object we have to check how much it was lowered. If it was lowered past some threshold it means that it is already in the hole and can be released. We have to make sure that the grip of the hand is tight enough so that the cube grasped will not rotate when placed partly above the hole.

This modification was implemented and successfully tried by R. Davis of the hand-eye group for other objects (wedge and slab) also. (See Figure 36).

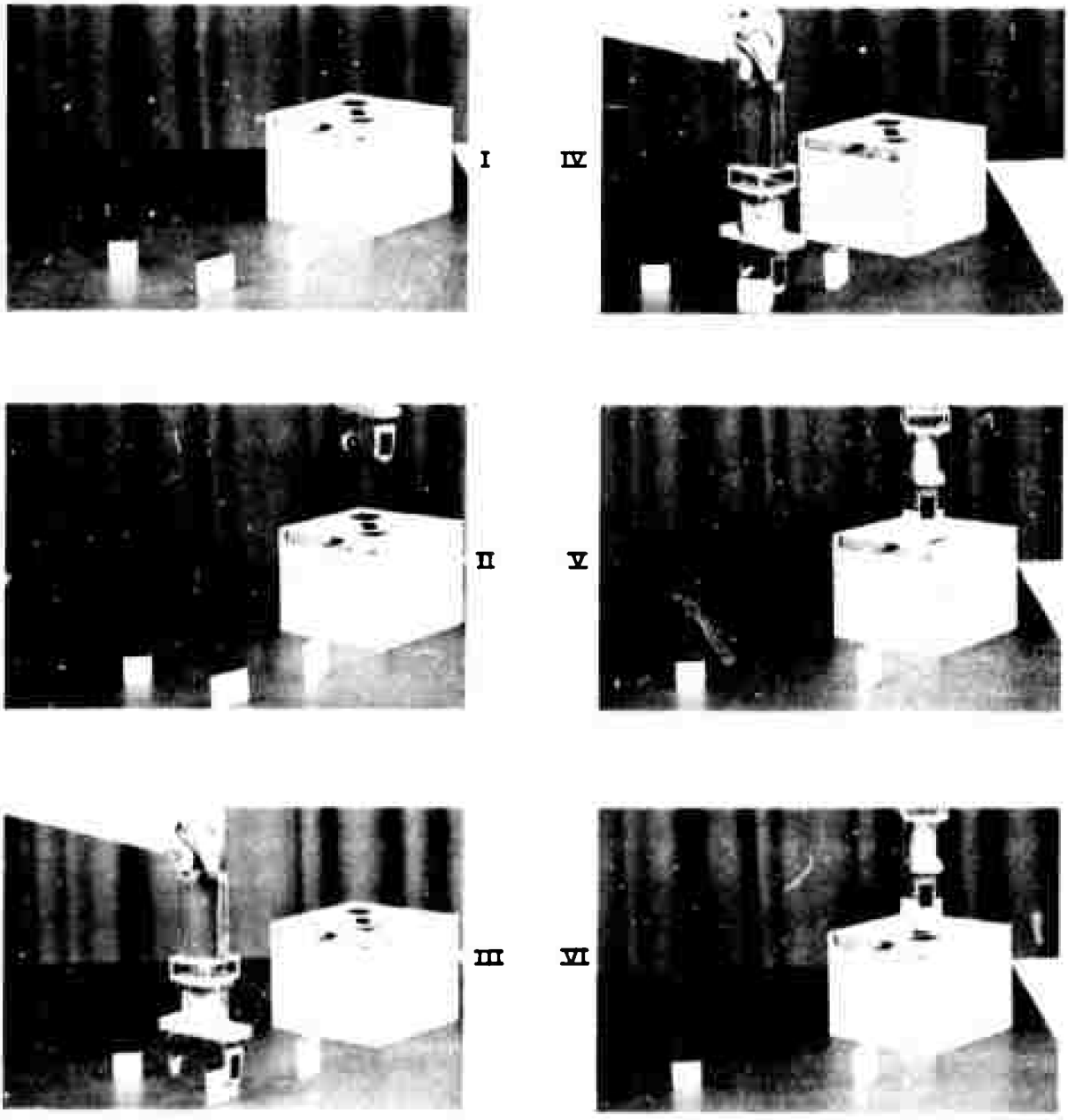


Fig. 36. INSERTING VARIOUS OBJECTS INTO HOLES.

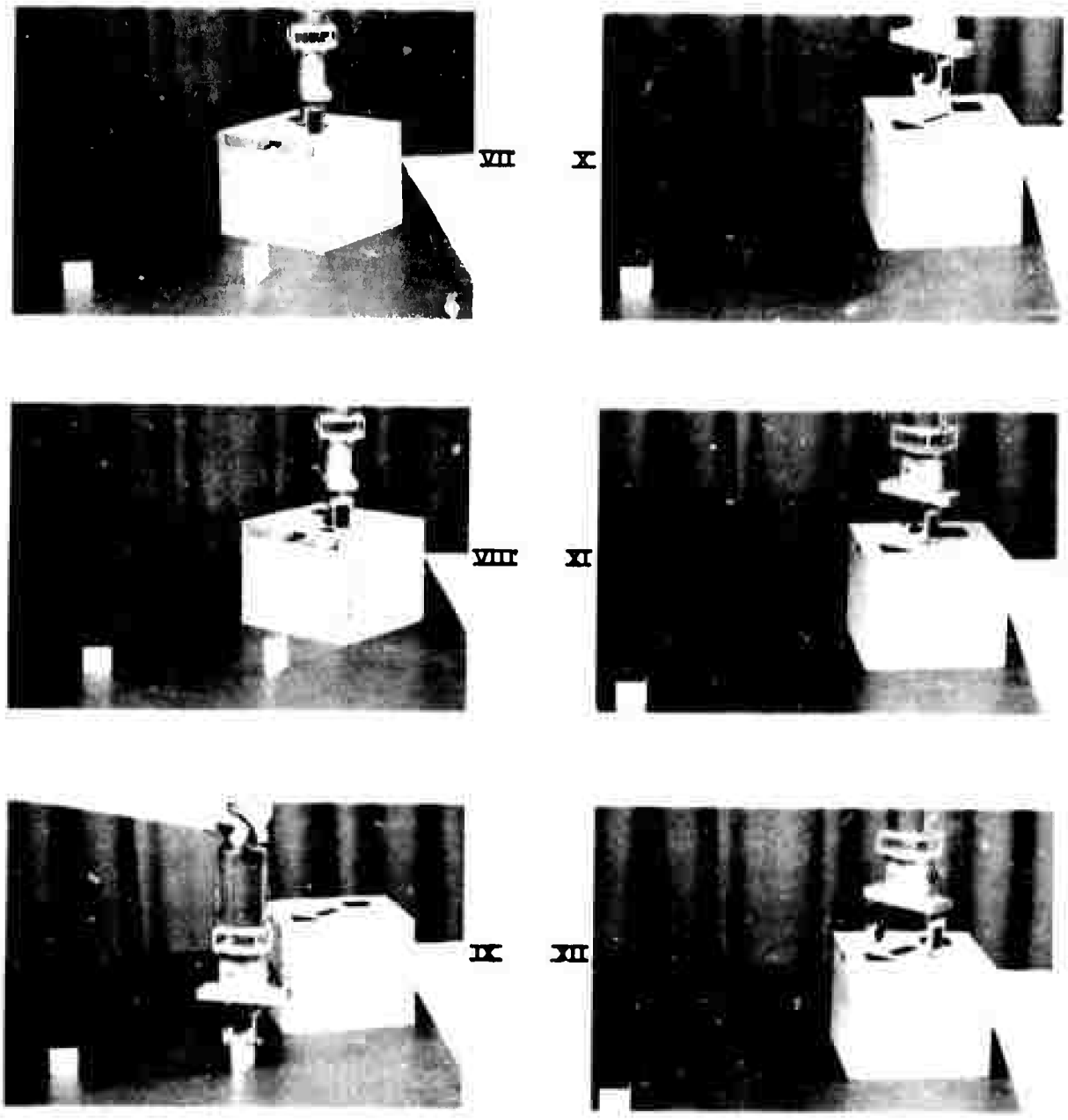


Fig. 36. CONTINUED.

CHAPTER 6: SUMMARY AND FUTURE WORK

6.1 Introduction

The actual results and performance evaluation of the three major components of this research, (corner-finder, calibration updating and visual feedback), are discussed in the next three sections. Since the performance of latter components relies on the performance of prior ones, these sections should be read in order. With the performance description we discuss the major shortcomings of the three components and their effects, and suggest possible future improvements or alternatives. Some of these have already been discussed in past chapters. In the last section we discuss possible directions for further research, to extend and generalize the results accomplished in this work.

Much effort went into making the programs effective and reliable. Each program can have, in general, five kinds of results: (a) It achieved the goal set to it; (b) It quit trying for justifiable reasons (in a sense, it took account of its own limitations); (c) It quit trying when there was no reason to do so (if it was a person we would say that he was lazy or not persistent enough); (d) It achieved the wrong goal but realized it (in this case there is some reason not to simply try again); (e) It achieved the wrong goal but did not realize it. In the last two cases we could also have undesirable side-effects like undoing a goal already achieved, loss of information, or even hardware damage.

By "effective" we mean that for the restrictive conditions under which we operate, the percentage of type (a) results will be high. By "reliable" we mean that the percentage of types (d) and (e) results will be very low and, when the side-effects include hardware damage, extremely low.

Each program was run several hundred times and in the case of the corner-finder several thousand. No formal statistics were gathered, for reasons that are discussed below, but estimated numbers for the effectiveness and reliability will be given.

There are two general types of causes for not achieving the results, after all the programming errors, user errors or hardware failures (Section 7.2.2) are either eliminated or discounted. One is excessive noise in the measurements of either the intensities of the image or various parameters of the system. The other is the particular details of a given situation, such as orientation of the objects, their colors, their relative position,

lighting conditions, etc, which make the task difficult. The effects of both causes are usually compounded.

The complexity of the situation and of the system makes it extremely difficult to analyze it theoretically, or to make statistically controlled experiments (of the Monte Carlo type for example). Therefore the only kind of numbers that we will cite are estimated from long experience with the system. Hopefully they are not biased by our wishes, or by the effect of unconsciously "doctoring" the conditions under which the experiments are executed, since, by properly (really improperly) choosing the conditions, a very high percentage of success could be achieved.

6.2 Corner-finder performance

While operating within the assumptions mentioned in Section 3.1.3, the effectivness of the corner-finder is estimated at more than 90%. When form and intensity matches are demanded the effectivness is even higher.

The program needs less than .5 seconds to analyze the regular 18*18 raster unit window (without recentering). Another .5 seconds is needed to display the information (intensity map, boundary points, fitted lines and features data structure) for the user when requested.

There are two situations which occur in the stacking task when some of the assumptions are violated and the performance is degraded. The first happens when two adjacent faces have very similar intensity. The other happens when the images of two faces are separated by a narrow strip (2 to 3 raster units wide) with different but somewhat widespread intensities. In both situations, sometimes, the histogram does not have the proper number of maxima.

The first situation is a tough nut for any scene analyzer. To find the boundary in this case, fine tuning of the sensitivity of the camera, and some kind of statistical analysis is needed. For example the line-verifier described by Tenenbaum [28]. The solution will be much simpler if the intensities are similar when viewed through the clear filter but actually the faces have different colors. If the colors are known to the program it can select the filter that will maximize contrast. If the colors are not known but difference of colors is suspected, the program can try the filters in turn till one (if any) gives enough contrast.

We tried to avoid the first situation by carefully painting the faces of the cubes in different shades of gray. Different magnitudes and directions of errors can cause different faces to meet. We need at least three differnt

intensities. Two of them (of the vertical faces) have to be different from the background intensity also. (See Figure 37).

Three alternatives are suggested to solve the problem of the second situation:

(a) A modified version of the corner-finder with histogram analysis tailored to the presence of narrow strips.
(b) An improved version of the operator described by Hueckel [12] that can detect narrow strips also, which is now being integrated into the hand-eye system.
(c) Use of a higher magnification lens which will cause the image of the strip to be wide enough for the corner-finder. The disadvantage of this solution is that more effort will be needed for centering and for focussing the camera, because now the field-of-view and depth-of-field are smaller.

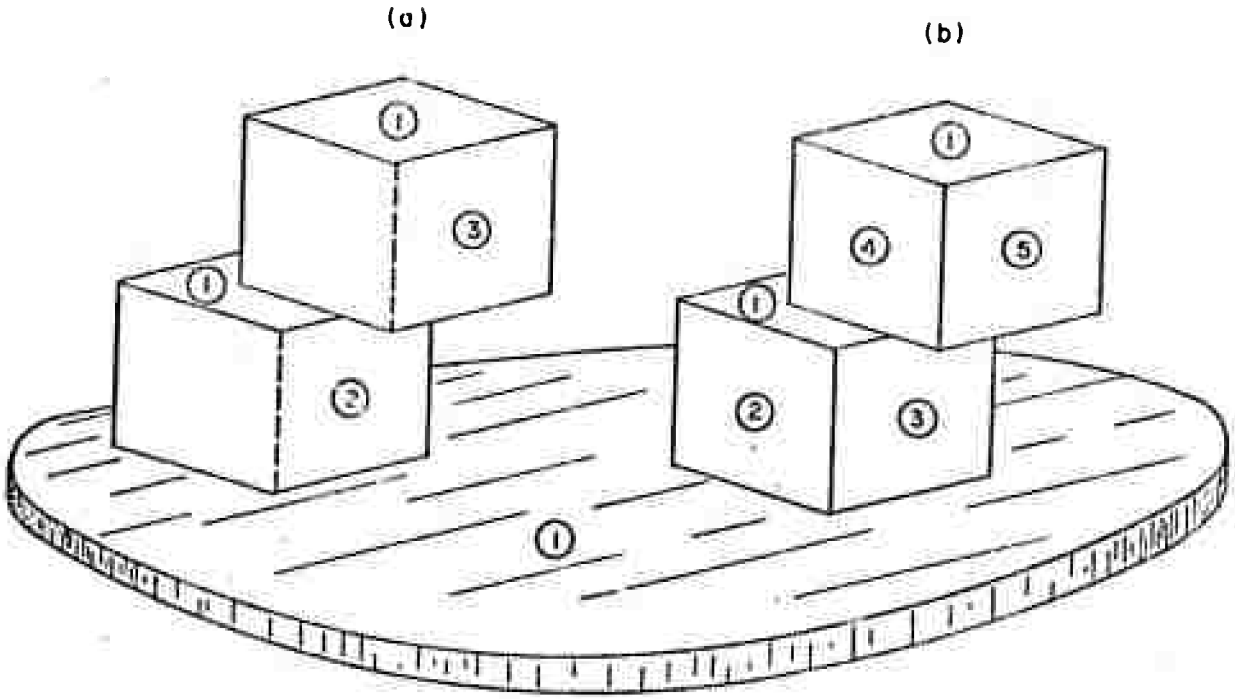
6.3 Calibration updating programs performance.

6.3.1 Pan/tilt calibration updating

When the degradation of coordination causes errors of less than 30 raster units on the image, or about 1 inch at typical operating range, the pan/tilt calibration updating from the arm will seldom fail.

The major reason for the effectiveness is that we can control most of the factors of the scene:

(a) The environment is simple and hence we can search.
(b) The hand is rotated so that the hand-mark will face the camera obscuring other features of the hand that are more complicated.
(c) The finger and hand-mark were painted to give high resolution.
(d) The dimensions of the hand-mark were chosen (constrained by the finger's size) to be compatible with the regular 18*18 raster units window and the 2 inch lens. The image of the hand mark is bigger than the window but the image of the finger beside the hand-mark is not too narrow.
(e) The form of the hand-mark (rectangle) was chosen so that the simple search algorithm already built into the corner-finder will lead to the right-hand lower corner whenever any part of the hand-mark is seen. (The reader can simulate this action, similar to the manner employed in Figure 23). If needed, the form of the hand-mark could be made more complex to avoid confusion with other objects.



(a) Arrangement of minimum number of intensities.

(b) Arrangement of preferred number of intensities.

Fig. 37.

PAINTING SCHEMES FOR VISUAL FEEDBACK TASKS FOR USE WITH THE CORNER-FINDER

In this case also, we can assure high effectivness by employing a more complex search algorithm tuned to the known form of the hand-mark.

Since the relative intensity and form are known, a form and intensity match is obviously demanded.

The first arm position used is the farthest from the camera so that coordination errors even larger than 1 inch can be tolerated. If the program succeeds in locating the hand-mark in the first arm position, the "instant adaptation" described in Section 4.4, and the fact that a large part of the error is deterministic in nature, will insure the program success at subsequent positions of the arm.

In a time interval of two weeks, degradation of coordination can cause averaged prediction error in the image of up to 20 raster units in the x-direction (see Figure 27), and 10 raster units in the y-direction. Occasionally (probably because of unauthorized tinkering), the errors are larger. One, or at most two runs of the calibration updating program will reduce the averaged error to less than 4 raster units. The standard deviation of the errors will be reduced from less than 6 to less than 3 raster units.

The CPU processing time needed for one run (13 positions of the arm) is about 1.25 minutes. More than half of this time is used for the real time control of the camera. A new camera, now being added to the hand-eye system, will move 5 times faster.

The effectiveness and processing time needed for calibration updating from cubes on the table do not make much sense since most of the work is done by the user, and hence will not be cited.

6.3.2 Focus calibration updating

Unlike pan and tilt, degradation in focus is rarely noticed and the program is run only after the camera is taken apart for repair or maintainance.

Using the automatic focussing routine is very time consuming because the focus is servoed 20 to 40 times per range point (we use 8 different ranges). Therefore the manual mode is used most of the time.

6.4 Visual feedback tasks performance

From long experience with the system, the

effectiveness of the grasping and placing tasks is estimated to be above 85%, while that of the stacking task is estimated to be about 70%. The failures of the stacking tasks are caused mainly by failures of the corner-finder. Reasons and possible improvements have already been discussed in Section 6.2.

The second cause of failure in the stacking task is movement of the bottom cube occurring when the top cube is placed on it. In similar cases we (persons) will use the other hand to hold the bottom cube so that it will not move. A similar solution should be investigated when the hand-eye system "grows" another arm. Till then (some months from now), we try to prevent movement by using heavy cubes, (which are needed also for proper functioning of the touch sensors), and a black rubber pad to cover the table. As an alternative we can try to locate the bottom cube again each time after the top cube was placed on it, if enough edges and/or corners of the bottom cube are not obscured.

The intricacies of the world model and its updating help to maintain the reliability at a very high level.

The coordination errors corrected by the visual feedback have magnitude up to .6 inches. If we take into account the maximum opening between the fingers (2.4 inches), their width (.75 inches) and the size of the cubes handled (1.25 and 1.50 inches), it can be seen that the magnitude of the error corrected is close to the magnitude of the errors that can be tolerated before the nature of the situation changes, especially in the grasping task.

If the magnitude of the coordination errors is very large, the hand can press the cube from above while trying to get over it (see Figure 38(a)), or the touch sensors, or even the whole fingers can miss the cube when closing. These situations are not handled currently. (See Figure 38(c) and (b)).

The first situation can be detected and corrected by finding the c.t.c. of the arm after it stopped because of the opposing force. The support hypothesis is used as external information. Unlike placing, the pressure in the case is less distributed and can cause movement of the cube. Because of the above reason we did not handle this case and decided to wait for more sensitive force sensing, or more touch sensor. (See Figure 38(d)).

The second situation can be detected as follows: If the coordination errors are very large, the fingers will miss the cube completely and close till they touch each other and then stop because a touch sensor was activated.

The opening between the fingers in this case is zero (see Figure 38(b)). If the errors are of lesser magnitude, the touch sensors will miss the cube but the rubber pads on the fingers will grip it (see Figure 38(c)). In this case the opening between the fingers will be almost the size of the cube but no touch will be indicated.

Finding the coordination error in the second situation is more difficult than in the first because we do not have any external information to use. In the first case we can find the direction of the error by observing if the cube obscures the hand or vice-versa, and correct by a cut-and-try method. The second case is even worse because the c.t.c. of the cube are lost by the sweeping motion of the fingers. This problem can also be solved by adding more touch sensors so that the fingers cannot touch anything without at least one touch sensor being activated (see Figure 38(d)).

Similar situations can occur in the stacking task.

Currently the incremental movement of the arm is realized by changing the angles of all six joints of the arm. The accuracy of the movement is limited by the precision of measuring the position of the first three joints (the "shoulder" joints). A design of a new hand is now in progress that will enable us to realize incremental motion of the hand in two perpendicular directions without moving the arm at all. It will be accomplished by connecting the hand to the arm with an extendible rod and by separate control of each finger. The positional errors in this case will be much less than before since the same relative error is now multiplied by a much shorter length.

6.5 Suggestion for future research

Each of the three research components suggests directions for future research to extend and generalize the results reported here:

(a) Corner-finder: Other scene analysis principles, (gradient following, region growing; use of color and texture, direct three dimensional measurement, etc.) should be adapted for use in visual feedback and similar applications, (i.e the programs using these principles should be able to use already known information at all levels of its operation).

(b) Calibration updating: The updating process should be extended to more parameters. For example: A white hand-mark could help to update the color calibration of the system when the lighting conditions are changing.

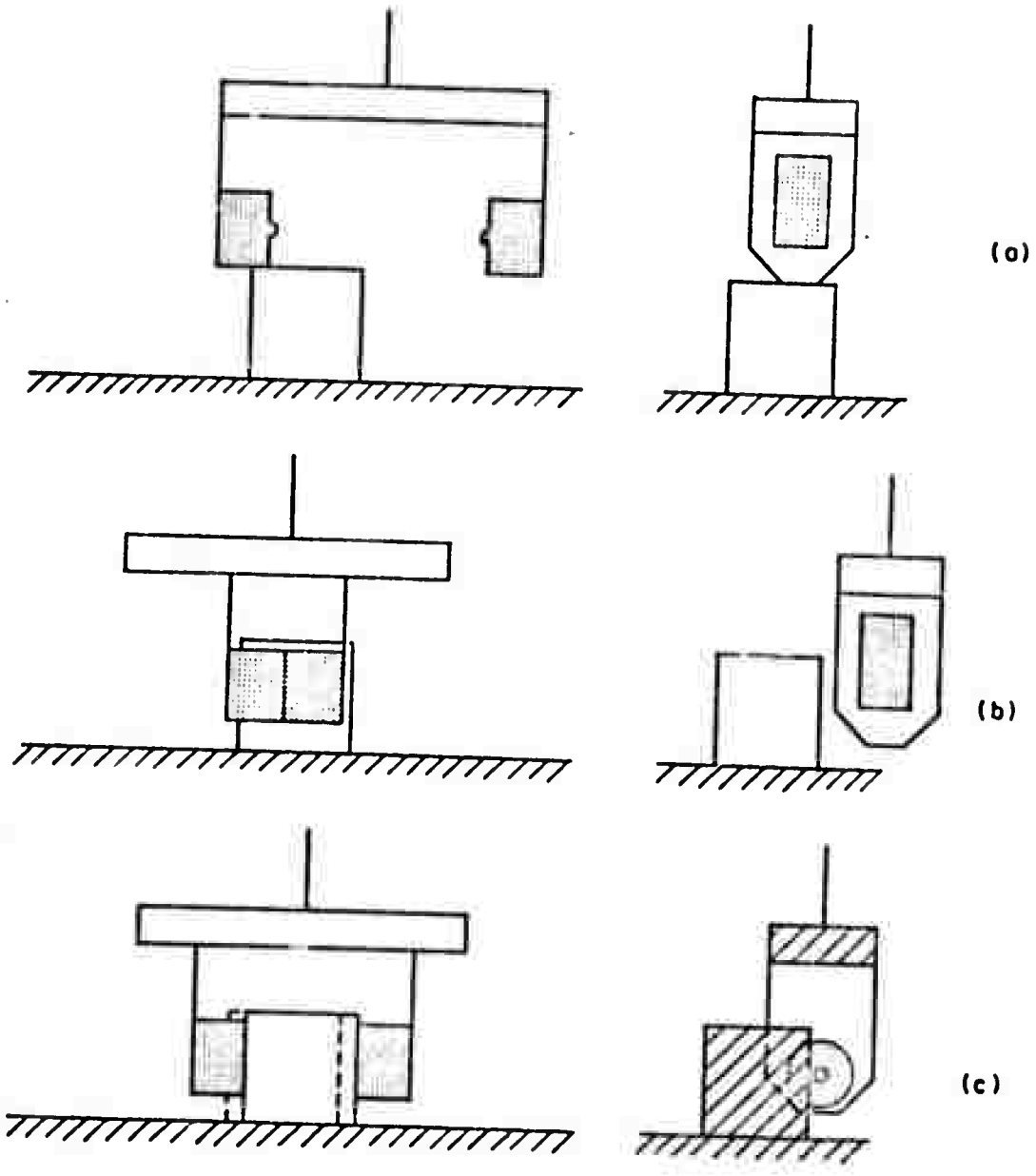
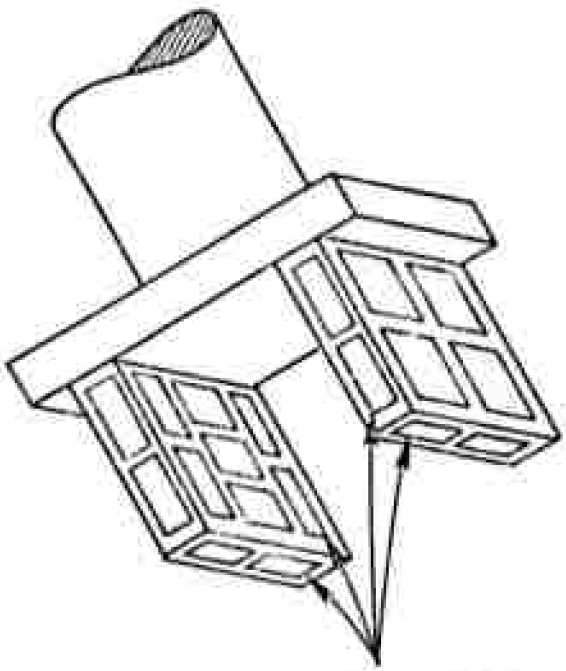


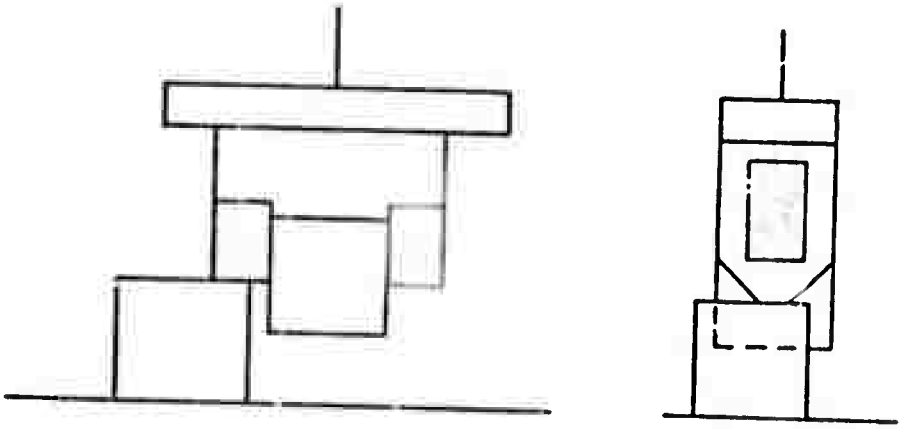
Fig. 38.

EFFECTS OF LARGE COORDINATION ERRORS IN THE GRASPING AND STACKING TASKS

(Continued on next page)



(d)



(e)

Fig. 38.

EFFECTS OF LARGE COORDINATION ERRORS IN THE GRASPING AND STACKING TASKS
(Continued)

Calibration updating using errors detected in regular operation of the system should be investigated including problems of utilizing errors detected with low confidence. Calibration updating of the arm can also be tried, especially after the system will have two cameras which can, in principle, define three dimensional coordinates without using the table, or the support hypothesis in general. The problem of correspondence is relatively easy to solve when only the hand is in view.

(c) Visual feedback tasks: The capabilities should be extended to handle more types of objects and relations between them. This means not just writing more programs to handle more situations, but rather generalizing the principles of operation to include more types of objects and relations.

We can now describe what might be the next steps of the development of the stacking task:

(a) A pair of edges are specified on the top face of the bottom object and another pair on the bottom face of the top object to be mated to the first pair or to satisfy some given relation. All four edges have to be visible.

(b) The top and bottom faces are specified with the relation they have to satisfy. In this case the program has to choose edges and relations to check.

The programs to accomplish the generalized task will need heuristics for automatic planning of the scene analysis needed for various manipulated objects and relations. The planning will have to take into account the fact that when the object to be moved can be grasped in more than one way (orientation), the program can choose the way which will make the scene analysis easiest. Currently the scene analysis and selection are written explicitly into the program for cubes.

In parallel, the subject of tactile feedback should be developed to use in situations where the view of the camera(s) is blocked, and to augment the visual feedback.

Some examples of possible augmentation were given in the last section.

In the next (and last) chapter we discuss some general problems that in my opinion are too important to gloss over or to put in an appendix.

CHAPTER 7: "GENERAL" PROBLEMS

7.1 Introduction

The purpose of this chapter is to bring to light a number of problems related to the design of robots or other large scale systems. Most of the problems are not new. They are brought up here because examples of them surfaced during the research. These examples are used to explain the nature of the general problems. For some of them a general principle of solution is suggested with a short description of the particular ways these solutions were implemented. For some of them no satisfactory solutions were found.

The problem and solution evolved in the following fashion:

particular problem --> general problem --> general principle of solution --> particular implementation of solution

Not all the problems are of the same importance as the reader can easily see.

7.2 Error sources and handling

7.2.1 General notes

The following characteristics of our system make the problem of error and failure handling both acute and of general interest:

(a) The programs have control of external hardware (camera, arm, etc.). The real time nature of the control is sometimes crucial (for example, the arm control takes the dynamics of the arm into account). The hardware does not have enough safeguards of its own because of design decisions, and not because of neglect. For example, there is no simple way to install mechanical stops or microswitches on the arm to prevent it from striking the camera.

(b) The collection of programs is very big (several hundred thousand words) and written by different people at different times.

(c) The main processor and some of the other devices are time shared between a number of unrelated jobs. This characteristic will still hold if the facilities are dedicated but the organization of the programs does not include an omniscient central controller.

In the following sections we will list possible error sources, the effects of errors and some problems and

solutions concerning the handling of errors and failures.

7.2.2 Error sources

The following are some of the common error sources:

(a) Programmer errors: This is the most usual kind, where the programmer mistyped or used the wrong equation. This kind of error is common during the development phase and hopefully eliminated from the experimental and operational phases.

(b) Program errors: Programs of course do not generate errors of their own, so that this kind of error can also be attributed to the programmer. They are more subtle and less explicit than those of kind (a). They occur when the complexity of the program causes a situation which the programmer did not think possible or did not realize could exist at all, and so did not make the right provisions in the program for the situation.

(c) Hardware errors: This type of error is inevitable. The frequency of their occurrence can be reduced by making the hardware more reliable. Elimination (i.e making the probability of occurrence extremely low) requires a high degree of redundancy (as exists for example in animals). This is too expensive in most cases. Hardware errors are caused either by malfunction or by noise (or in general by some parameter getting out of its specified range).

(d) User errors: During the development phase, and after, the programs interact with the user. By accident, or ignorance (or malice), the user can feed the program incorrect information.

7.2.3 Effects of errors

The effects of errors range from task incompletion through the anti-social tying up of the processor, to the major catastrophe of hardware destruction or even human injury. We would like to avoid the first two and completely eliminate the third if possible.

7.2.4 Error detection

The following considerations affect the design of error detection:

(a) To insure against hardware malfunction or destruction we would like the routines which actually operate the hardware (such as the arm and camera servo

programs) to check all the parameters given to them,

(b) Leaving the error checking till the last possible moment is uneconomic in the sense that we would save much time wasted in calculations that become useless when the error is detected. Hence we would like to detect the error as soon after its generation as possible. However, the determination where a chain of computation started with a parameter that eventually will generate an error is very difficult.

In the programs described here, all the routines which communicate with the hardware were programmed to check the parameters given to them. Detection of errors at an earlier stage was left optional.

7.2.5 Errors and failures handling

Detected errors are handled like failures. Failures are situations where a routine cannot complete its task or function. Unlike errors, failures happen naturally in any program that uses search or cut-and-try methods. Sometimes the failure of a subroutine means success of the calling routine if it wanted to refute a hypothesis that the subroutine was asked to check. For example: If we want to prove that a given space on the table top is empty, we call the corner-finder and ask it to find any feature there. If the subroutine SLICE (see Section 3.2.2.) reports a failure (i.e it found that the intensity in the window is uniform or that the intensity histogram has only one maximum), then we have succeeded in proving that the space was indeed empty.

Error and failure handling poses the following general problem that I will try to explain with an example. Let us say that we have a routine R1 that can terminate either with success or with failure. Next we design a routine R2 which calls R1. After R1 is called by R2 and executed, R2 checks the termination condition of R1. If R1 failed, R2 will then execute some action A2. Later we design a routine R3 which also calls R1. Upon failure of R1, R3 executes A3. So far so good, but now comes the trouble. We now want to design two more routines R4 and R5 each of them calling R2. But, R4 would like R2 to carry out action A4 if R1 fails, and R5 would like R2 to carry out action A5 if R1 fails (A4 and A5 are different from each other and both might be different from the original A2). If the nested calling would stop here we might go back and change R2 so that it would know whether it was called by R4, or by R5, or by neither, and carry out the appropriate action. But the design process goes on, and on, and R6, R7 are designed each calling R2, R3, R4, R5 at various stages and so on.

An alternative solution is to design a super-duper error and failure handling routine which will be called each time an error is detected or a failure is indicated (from now on we will refer only to failures but we will mean both). This routine will have access to the nested calls stack and in general to the state of the process. It would then fix the state as best it could and transfer control appropriately. In simple cases it would return control to the interrupted routine.

The advantage of this scheme is that errors and failures are dealt with on the spot. Also no redesign of the routines is needed each time a new subroutine which causes them to be called is added. The disadvantage is complexity since the error handling routine has to provide for all contingencies. Finding all the contingencies is not a trivial problem. Also, the error handler will have to be changed, or at least added to, each time a new routine is designed. The transfer of control needed, and the ability to interrogate the full state of the program, cannot be implemented in SAIL (and similar languages) in a straightforward manner. It could be done in the improved version of SAIL now being designed.

In our case the disadvantages were prohibitive and we used the first scheme (mentioned at the end of the above example), without much redesign. In the terms of the last example, we would not redesign R2 after designing R4 and R5. What would then happen is the following: failure of R1 when called by R2 which in turn was called by R4 (or R5) would cause R2 to take action A2, but since this action is inappropriate it would cause failure of R2 also. Then R4 (or R5) can deal differently with the failure, albeit a little late (it could be worse if there were more nested calls levels). The information that R1 is the real cause of the failure can be (and is) filtered up to R4 and R5 (and if needed even higher in the hierarchy). Note that leaving the error handling to higher level routines has the advantage that these routines will have more general knowledge, but on the other hand this knowledge can be too condensed and not detailed enough to handle the error efficiently.

There is no simple solution to this problem. A very complicated answer might be planning. When the system is given a goal it will plan a series of subgoals, checks, branches and loops leading to this goal. This gross planning will not be detailed enough to handle all errors or failures and hence each subgoal has to be analyzed and planned for. This should be repeated for lower levels till we reach the level where the built-in failure handling is reliable enough. This planning for errors and failures is analogous to the automatic writing, specifically for each goal and

tailored to the contingencies anticipated, of a super-duper error handling program, similar to the one mentioned above.

This scheme is somewhat wasteful and we would like to be able to "store" the detailed plans. Since each goal is at least a little different from preceding goals, a "generalized" but detailed (see the contradiction?) version of the plans should be stored. Then we will use an execution monitor that will fill in the missing details on the go and without repeating most of the effort that went into planning. Work in a similar direction is in progress at S.R.I [15].

7.3 Empirical parameters

Many of the programs developed through this work, especially those dealing with image processing, have many parameters, mostly in the form of thresholds which are empirically fixed at a certain value. During the program development, no clear method for the determination of their optimal values, for various anticipated situations, was found that could then be given to a program. Hence they were fixed at some value thought to be optimal for most situations. They are too numerous to be passed as formal parameters and, because of the reason stated above, too confusing to be treated as such. On the other hand we would like to make them visible to a future user who will have situations not anticipated at design time which need either a change in the fixed optimal value or a variable value for which an algorithm can be found. No good solution was found to this problem apart from giving a list of them with enough explanations in the program documentation.

7.4 Guide-lines for program writing

The size of the programs developed and used in the hand-eye project and the fact that they are (and were) written by a number of different people over a long period of time, makes a set of guide-lines (if not enforced conventions) for program writing a necessity.

The following is a long list of parameters that should be optimized, or at least considered, when writing a program:

(a) Small size of source code.
(b) Small size of object code.
(c) Small core image at execution time.
(d) Fast compilation.
(e) Fast execution.

(f) Readability of the source code especially by other users. This will also ease modifying the code when needed.

(g) Modularity: General utility procedures and others should be written so that they can be cleanly lifted out of one program and transplanted into another. This means that the procedure should use formal parameters and not global variables and that it should make explicit reference to any subroutine it uses which was defined outside it.

(h) Versatility: The procedures should be applicable under a broad range of conditions without failure or noticeable degradation. They should not be too versatile if this necessitates complex settings of flags and parameters.

(i) For debugging and experimental purposes, the programs need conversational ability so that parameters and even structure can be modified by the user without recompilation. However, after the reliability of the program has been satisfactorily proved, most of this code should be taken out in an orderly fashion.

Trying to optimize all the above parameters is working at cross purposes. One trivial example: The use of a table instead of computation to find values of functions will speed up execution but also increase needed storage. In the programs developed in this research, optimizing parameters (f) to (i) had more importance than optimizing parameters (a) to (e). I would like to have a compiler with some flexibility as to what parameters it tries to optimize.

7.5 Module organization

Currently we have about ten modules apart from drivers. The hand-eye monitor and the way messages are handled allow complete flexibility as to which module controls what module. After one module activates a message procedure in another module it can either wait till the other module reports termination of the called procedure, or go ahead and continue independent execution. There is no guaranty that the called procedure will ever return control to the calling module.

It is up to the programmer to build some structure in the control of modules using the flexibility mentioned above. Some classes of structures are discussed below.

(a) "Prussian army" or "Tree" structure (See Figure 39(a)):

Each subtree can perform a specific task. The tasks become more simple and specific the lower they are in the tree. Each subtree is "rugged" in the sense that it can perform its task without much help from above. Commands go down the tree and results are reported upward. Communication

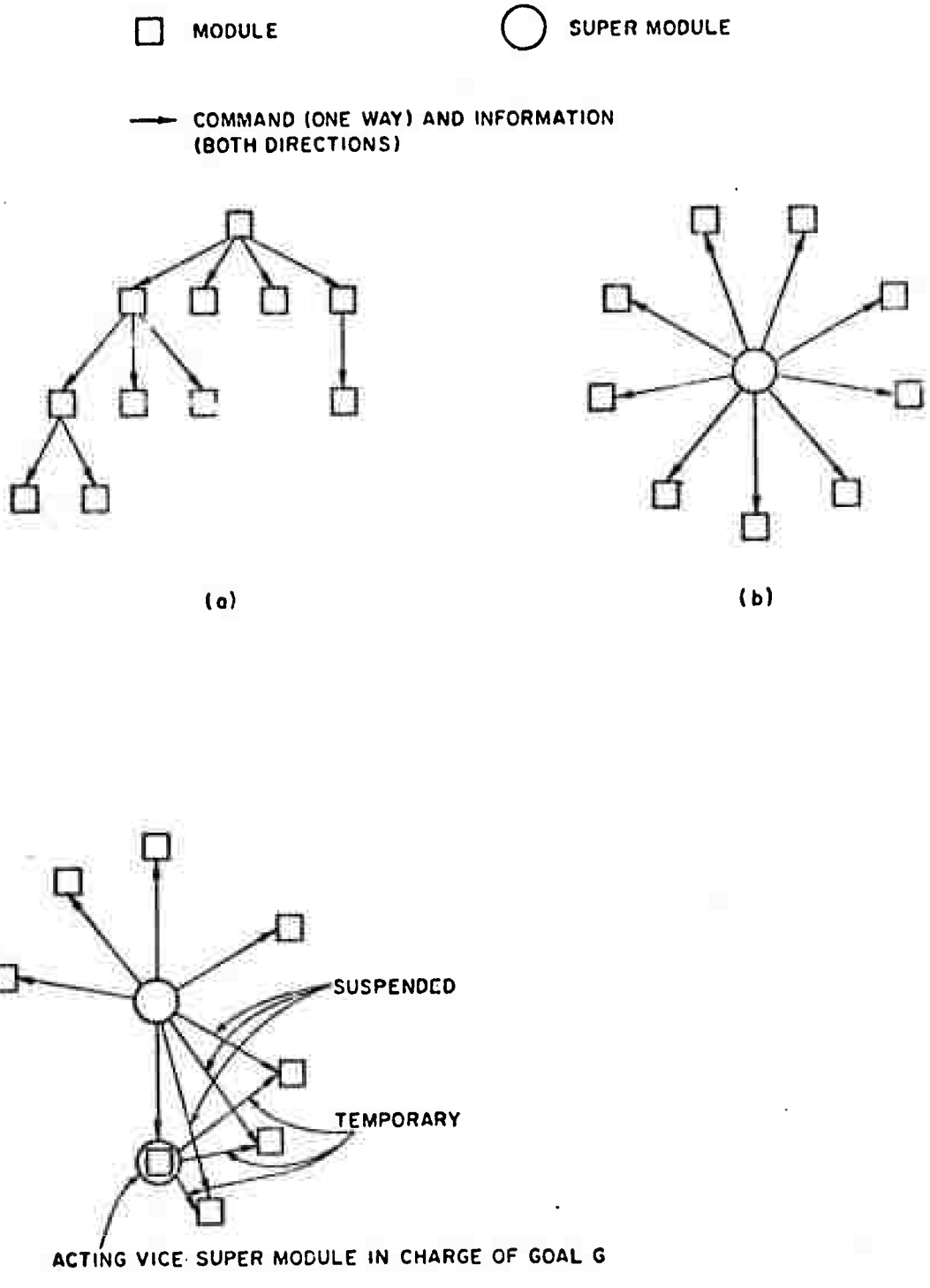
between modules is done only through a common ancestor (or common commander). This structure is rigid but simple and suitable for a development effort which does not have to be highly coordinated. The development can start from both ends of the tree at the same time. The rigidity and simplicity make the solution of the optimization of resources allocation very difficult. Another disadvantage is that many of the "soldiers" have very similar tasks but they cannot be used outside their "units". A variant of this structure is discussed later.

(b) "Star" structure (See Figure 39(b)):
One super-driver is controlling all modules. All the communications between modules are done through the super-driver. The super-driver will be very complex but can make efficient use of the modules and other resources. All the modules have to be designed together, otherwise changes (sometimes extensive), will be needed every time a new module is added or an old module is changed, since for efficient planning the super-driver needs to know the exact characteristics of the modules.

(c) "Spoked wheel" structure (See Figure 39(c)):
Again we have a super-driver controlling all modules. But in addition there are channels of communication and, what is more important, channels of control between the modules, governed by the super-driver. With those means the driver can give one module the ability to control other modules for the purpose of achieving a certain subgoal. In this manner the "knowledge" needed is diffused but not to the extent of the tree structure.

The organization structure used for the system described here is a variant of the tree structure, where the lower level modules are more generalized and thus can be used by more than one module. (See Figure 2). Procedures in the generalized lower level module are activated by messages sent to it by different modules. The message handler puts these messages in a separate queue for each module to wait for the termination of all activities required by prior messages in the queue. Alternative ways would be to use identical copies of the generalized lower level procedures, or to make them reentrant.

The main conclusion to be drawn from this chapter and from the thesis in general is that things are always more complicated than they first look. In fact, the entire field of Robotics has been concerned with solving problems which appear at first to be trivial.



(a) "Prussian Army"

(b) "Star"

(c) "Spoked Wheel"

Fig. 39.

EXAMPLES OF MODULE ORGANIZATION STRUCTURE

Appendix A: Camera module

A.1 Camera module message procedures

The camera module is activated by the following messages:

CAM_INIT

Reads the camera model parameters from disc file.

CAM_UPDATE

Reads the pan tilt and focus pots, computes the camera transformation (see Section A.2) and stores it in global CAMERA_MODEL. Since the pot readings are noisy because of power supply noise, A/D converter noise, high frequency mechanical vibrations of camera parts relative to each other, etc., the pots are read a number of times (40) and the readings are averaged. As an indication of the amount of noise left in the averaged readings we use the following expression:

(MAX-MIN)/[4*SQRT(N)]

Where: MAX is the highest reading, MIN is the lowest reading, and N is the number of readings. The noise is checked to be less than that corresponding to 1/20 of depth-of-field in focus, 10↑(-4) rads. in tilt and 2.5*10↑(-4) rads. in pan.

CHNG_LENS(N)

Rotates the turret till lens no. N is facing the vidicon. It takes about 1 sec. to rotate the turret to the next position.

MOVE_CAM(PAN,TILT)

Moves the pan/tilt head so that the pan and tilt angles in radians equal to PAN and TILT, respectively. The velocity of the tilt movement is .06 rads/sec. and that of pan .13 rads/sec.

CHNG_FOCUS(RANG)

Moves the vidicon to focus at a range of RANG inches from the lens center. The velocity of the vidicon movement is .012 inch/sec. For example: to change the distance at which the camera is focussed from 20 inches to 50 inches with the 2 inch focal length lens takes 11.5 sec.

CAM_CENTER(N,X,Y,Z)

Computes the pan and tilt angles necessary for the principal ray of the camera to pass through the point (X,Y,Z) in t.c.s. (i.e that the image of this point will be at point (PP1,PP2) in the i.c.s.), when lens no. N is used. (See Section A,3). It also computes the range from the lens center to the point. Then the camera is servoed to the computed setting and the lens is changed accordingly.

The above four message procedures use a common servo routine in the following way: using the pot constants and the focus equation, the pan and tilt angles and the range are converted to pot settings. These settings are then checked for legality. The legality of the lens number given to it is also checked. Each 1/60 sec the readings of the pots are compared to those required. If the differences are larger than the given thresholds, the motors are actuated in the proper direction and run at the constant rates mentioned above. The thresholds (full) correspond to 1/5 depth-of-field for focus and 3 mrads. in pan and tilt. Note that these thresholds are much higher than the noise requirements mentioned in the discussion of CAM_UPDATE. The reason is that centering is done to assure that the object will be approximately in the middle of the field of view, where the performance of the camera and the model are best. On the other hand, to get precise measurements we need to know where the camera is pointing with high precision. After the motion is completed or aborted CAM_UPDATE is called automatically.

CAM_PRED(N,X,Y,Z)

Computes PAN, TILT and RANG as in CAM_CENTER, and then computes the corresponding camera transformation and stores it in global CAMERA_PRED. No servoing is done.

The output of the camera module is as follows (all global variables): CAMFLG will be nonzero if an error occurred during the execution of the procedure called, and its value gives the cause of the error. CAMPAN, CAMTIL and CAMRANG hold the current pan and tilt angles and the range at which the camera is focussed. CAMLENS designates the current lens. PANPOT, TILPOT and FOCPOT hold the current averaged readings of the pots. CAMERA_MODEL and CAMERA-PRED hold the transformation of the current camera position and of the position mentioned last in CAM_PRED.

A.2 Camera transformation matrix computation

Take a point B with coordinates [B]=(xt,yt,zt) in the t.c.s. Its coordinates in a c.s. parallel to the t.c.s. and translated to the point P with coordinates

[P]=(P1,P2,P3) in the t.o.s., will be:[B]-[P], P is the point of intersection of the pan and tilt axes. Its coordinates in a c.s. rotated relative to the last one with Euler angles PSI, THETA and PHI (right hand rotation in this order) will be:

[R1(PSI,THETA,PHI)]*([B]-[P]).

where R1 is an orthonormal rotation matrix. The three Euler angles are given in terms of our conventions for the pan and tilt angles PAN and TILT, and the fixed rotation angle SWING as (see Figure 26):

PSI=PAN+π; THETA=TILT; PHI=SWING;

We will denote the rotation matrix expressed in terms of PAN, TILT and SWING by [R2], i.e:

[R2(PAN,TILT,SWING)]=[R1(PSI,TETA,PHI)].

Hence the coordinates of the point B in the last coordinate system will be: [R2]*([B]-[P]). We now rotate the c.s. with Euler angles -π/2, 0, -π/2. The coordinates of the point B in the new c.s. will be:

[R1(-π/2,0,-π/2)]*[R2]([B]-[P])

We define a matrix [R] by: [R]=[R1(-π/2,0,-π/2)]*[R2]. The coordinates of a point B in the last c.s. will be: [R]*([B]-[P]). Lastly, we translate the c.s. to a point C with coordinates [DP]=(DP1,DP2,DP3) in the last c.s. to get the "camera coordinate system" (c.c.s.). The coordinates of the point B in the c.c.s. which will be denoted by [U]=(u,v,w) are:

[U]=[R]*([B]-[P])-[DP]

For the sake of completeness, the elements of the matrix [R] which will be denoted by RIJ (e.g. R12) will be given here explicitly in terms of PAN, TILT and SWING, which will be abbreviated by P,T and S respectively:

```
R11=-COS(S)*SIN(P)+SIN(S)*SIN(T)*COS(P)
R12= COS(S)*COS(P)+SIN(S)*SIN(T)*SIN(P)
R13=-SIN(S)*COS(T)
R21= SIN(S)*SIN(P)+COS(S)*SIN(T)*COS(P)
R22=-SIN(S)*COS(P)+COS(S)*SIN(T)*SIN(P)
R23=-COS(S)*COS(T)
R31=-COS(T)*COS(P)
R32=-COS(T)*SIN(P)
R33=-SIN(T)
```

The pan and tilt angles are computed from the

corresponding pot readings as follows:

```
PAN=PPOT0+PPOTD*PANPOT;
TILT=TPOT0+TPOTD*TILPOT;
```

As mentioned in Section 4.2.2 SWING is an invariant parameter.

Define a vector [M]=(M1,M2,M3) by: [M]=[R]*[P]+[OP], now define a 3*3 matrix [EXT] as follows:

```
EXT11=R11; EXT12=R12; EXT13=-M1;
EXT21=R21; EXT22=R22; EXT23=-M2;
EXT31=R31; EXT32=R32; EXT33=-M3;
```

The matrix [EXT] can be used to convert coordinates of points on table-top from t.c.s. to c.c.s. in the following manner: For a point B on the table with coordinates (xt,yt,0) form a vector [T]=(xt,yt,1). Its coordinates in the c.c.s. are given by: [U]=[EXT]*[T].

To convert the coordinates of a point in c.c.s. denoted by [U]=(u,v,w), to its coordinates in i.c.s. denoted by (xi,yi) we use the 3*3 matrix [INT] defined as follows (see Figure 27):

```
INT11=FOC*KX;      INT12=0;        INT13=PP1;
INT21=-FOC*KX/333; INT23=FOC*KY;   INT23=PP2-PP1/333
INT31=0;           INT32=0;        INT33=1;
```

The distance between the vidicon face and the lens center is computed from the reading of the focus pot as follows:

```
FOC=FOCLEN0+FOCLENG*FOCPOT;
```

The matrix [INT] is used in the following way: compute a vector [J]=(J1,J2,J3) by: [J]=[INT]*[U], then, xi=J1/J3 and yi=J2/J3;

Last, we define a 3*3 matrix [COL]=[INT]*[EXT]. This matrix incorporates all the parameters of the camera model and its use should be obvious from the above, but we will repeat it anyway. To convert the coordinates of a point on a table given by its coordinates in t.c.s. [B]=(xt,yt,0) to its coordinates in the i.c.s. (xi,yi), we form a vector [T]=(xt,yt,1). compute [J]=[COL]*[T], and xi=J1/J3; yi=J2/J3.

If we want to compute the image point corresponding to a point not on the table-top, we use the lens center whose coordinates in the t.c.s. are given by

[C]=[P]+INV([R])*[DP]. We compute the line through the given point and the lens center, then we compute the point of intersection between this line and the table-top and proceed as before.

To compute the point of intersection between the table-top and a line through the lens center and an image point with coordinates (xi,yi) in the i.c.s., we form the vector [I]=(xi,yi,1), compute [J]=INV([COL])*[I], and xt=J1/J3; yt=J2/J3.

The matrix [COL] is called the camera transformation matrix. It is computed and stored each time when the camera module executes CAM_UPDATE or CAM_PRED.

A.3 Derivation of camera centering equations

To center the camera on a point defined by its coordinates in the t.c.s. [B]=(xt,yt,zt), we compute its coordinates in the i.c.s.:

[U]=(u,v,w); [U]=[R]*([P]-[B])-[DP];

Then we compute the needed pan and tilt angles so that u=v=0; To simplify the resulting equations we assume that SWING=0. This assumption is justified since the calibrated value of SWING is less than 2 deg. With this assumption we have:

u= (P1-xt)*SIN(P)-(P2-yt)*COS(P)-DP1
v=-[(P1-xt)*COS(P)+(P2-yt)*SIN(P)]*SIN(T)+(P3-zt)*COS(T)-DP2
w= [(P1-xt)*COS(P)+(P2-yt)*SIN(P)]*COS(T)+(P3-zt)*SIN(T)-DP3

First we solve u=0 for P (we denote the solution by PC). Then we substitute this value into the equation: v=0, and solve it for T (we denote the solution by TC). These two equations are solved by converting them to quadratic equations in COS(P) and COS(T) respectively. Last, we substitute PC and TC into the expression for w to get the distance between the lens center and the point [B]. We denote this distance by WC. Then we use the focus equation to compute the corresponding foo.

A.4 Derivation of pan/tilt updating equations

The goal of the pan/tilt updating routine is to update or recompute the pan and tilt pots parameters. It is done in the following way: For a point given by its coordinates in the t.c.s. [B]=(xt,yt,zt) the routine finds the corresponding image coordinates (xi,yi). From this

information we would like to compute the pan and tilt angles of the camera. These angles with the corresponding pot readings will enable us to recompute the pot parameters.

We define: $\alpha=u/w$; $\beta=v/w$. From the equations of the internal model (see Section A.2) we have:

$$\alpha=(x_i-PP1)/(foc*KX);$$
$$\beta=(y_i-PP2)/(foc*KY)+(x_i-PP1)/(333*foc*KY);$$

Using again the assumption that SWING=0, we get from the expressions for u,v and w given in the last section the following two equations which we have to solve for the pan and tilt angles P and T:

$$(P1-xt)*SIN(P)-(P2-yt)*COS(P)-DP1=$$
$$\alpha*([(P1-xt)*COS(P)+(P2-yt)*SIN(P)]*COS(T)+(P3-zt)*SIN(T)-DP3)$$

$$-[(P1-xt)*COS(P)+(P2-yt)*SIN(P)]*SIN(T)+(P3-zt)*COS(T)-DP2=$$
$$\beta*([(P1-xt)*COS(P)+(P2-yt)*SIN(P)]*COS(T)+(P3-zt)*SIN(T)-DP3)$$

I could not find an explicit analytic solution to the above equations, but the following approximate solution proved itself to be appropriate.

We denote by PC and TC the pan and tilt angles which solve the centering equations of the last section. We now define PD and TD by: PD=P-PC and TD=T-TC. Note that for $\alpha=\beta=0$, PC and TC are actually the solutions of the above equations and hence PD and TD are also equal to 0 in this case. Since in the updating process we centered the camera before finding xi and yi and since we assume that the calibration had not been degraded too much, we can assume that $u,v<w$ and hence $\alpha,\beta<1$ and PD,TD<1 or:

$$SIN(PD)=PD;\ SIN(TD)=TD;\ COS(PD)=COS(TD)=1;$$

With these assumptions we get the following equations for PD and TD:

$$[G+\alpha*DP1*COS(TC)]*PD-[\alpha*DP2]*TD-[\alpha*DP1*SIN(TC)]*PD*TD=\alpha*WC$$
$$DP1*[SIN(TC)+\beta*COS(TC)]*PD-[WC+DP3+\beta*DP2]*TD$$
$$+DP1*[COS(TC)-\beta*SIN(TC)]*PD*TD=\beta*WC$$

where G is defined by: $G=(P1-xt)*COS(PC)+(P2-yt)*SIN(PC)$. DP1,DP2,DP3 are of the same order of WC and somewhat smaller, assuming that the camera is not looking vertically down (which is always true with the camera currently used) we can say the same about G. With this assumption and the assumptions mentioned above we can simplify the equations to read:

$$G*PD-\alpha*DP2*TD=\alpha*WC$$

$$DP1*[SIN(TC)+\beta*COS(TC)]*PD-[WC+DP3]*TD=\beta*WC$$

Solving the two linear equations and using the same assumptions again we have:

$$PD=\alpha*WC/G$$
$$TD=-WC*[\beta-\alpha*Dp1*SIN(TC)/G]/(WC+DP3)$$

Appendix B: Automatic focus module

The automatic focussing routine was suggested and experimentally implemented by J.M Tenenbaum, and then modified by me for use in the focus pot calibration updating program and similar applications (focussing on an object at unknown range seen as a "blob" in the image before analyzing the image). It is too crude, as it is, to find range differences between two points closer than 1 inch (at 30 inches the depth of field of the 2 inch lens is .6 inch).

The principle of operation is to search for a range by moving the vidicon relative to the lens, at which a score function, which measures the the quality of focus, is maximized.

The module uses a window defined externally in LOOK_AT. The block diagram is given in Figure 4B. Each of the blocks exit and failure is indicated (in global FOCFLG) if either INPUT (see Section 3.2.2) or the camera module have failed. Other exit conditions are mentioned in the block descriptions which follow.

INIT

The camera is focussed at the current estimate of the range (the initial estimate is given as a parameter). SETCLIP is called to set the clips to bracket the intensity range seen in the window at that focus setting.

The next search interval is selected to be the larger of: (a) the current estimate of the range uncertainty (the initial estimate is given as a parameter),and (b) 3*DOFREQ where DOFREQ is the minimimum attainable uncertainty which is defined as the depth-of-field. DOFREQ is computed by an approximation to a more complex expression as follow:

DOFREQ=(R↑2)*C*(F#)/(F↑2).

R-is the range (the current estimate is used); F-is the lens focal length; F#-is the aperture setting. Since the aperture setting is not computer controlled it is left at maximum opening (1.4 for all lenses); C-is the dimension of a raster cell of the vidicon (approximately 1/600 inch). For the above formula we assumed that the "circle of confusion" has a diameter of half a raster unit.

The number 3 in the above equation was empirically chosen. It was found experimentally that the criterion used to judge the quality of focussing (see below) has values above 90% of the maximum in an interval of 3*DOFREQ around

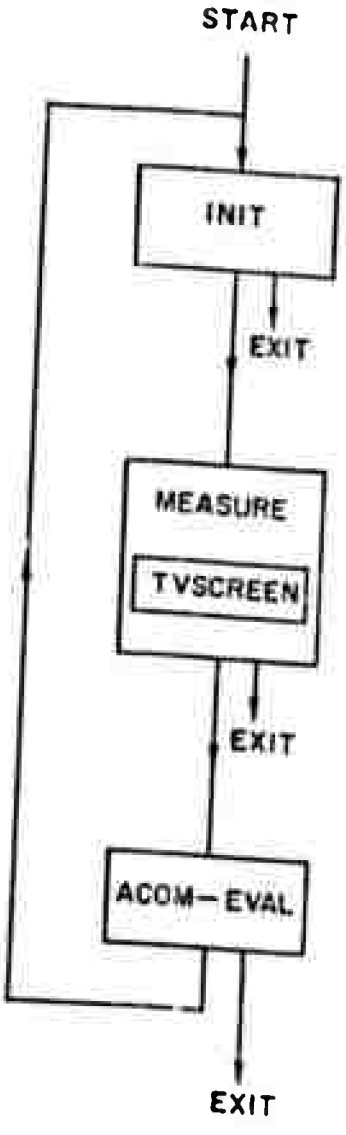


Fig. 4c.

BLOCK-DIAGRAM OF THE AUTOMATIC FOCUSSING MODULE

the maximum.

The search interval is centered on the current range estimate.

MEASURE

10 equidistant points are picked in the search interval. At each point a score function (criterion) is computed by TVSCREEN (see below). The three highest scores are dynamically maintained with their corresponding ranges.

TVSCREEN

The window is read in 4 times and the time-average intensity is computed at each sampling point. Using the averaged intensities, the magnitude of the gradient at each sampling point (less those lying on the perimeter of the window) is computed using the following approximation:

```
G(X,Y)=SQRT([I(X-1,Y-1)+2*I(X-1,Y)+I(X-1,Y+1)
     -I(X+1,Y-1)-2*I(X+1,Y)-I(X+1,Y+1)]↑2+
     [I(X-1,Y-1)+2*I(X,Y-1)+I(X+1,Y-1)
     -I(X-1,Y+1)-2*I(X,Y+1)-I(X+1,Y+1)]↑2)
```

where I(x,y) is the averaged intensity at the sampling point (x,y). From G(x,y) we compute a thresholded value of the gradient's magnitude as follows:

```
H(X,Y)= G(X,Y)    If G(X,Y)≥CUT
      = 0         otherwise
```

CUT is a parameter given to the program or computed in INIT to be a fraction of the maximum of H(X,Y) over the window at the point of highest score. The score function is the sum of H(X,Y) over the window.

ACOMEVAL

Using the 3 highest scores and the corresponding ranges, the new range and range-uncertainty estimates are computed. Then a decision is made to exit or enter MEASURE again according to the following rules:

(a) If all the scores are 0 then ACOMEVAL exits and failure is indicated.

(b) If the range corresponding to the highest score point is too close to the search interval boundary points then the range of this point is the new estimate, the range uncertainty estimate is unchanged and MEASURE is entered again.

(c) If the new uncertainty estimate is larger than

the old one, or if it is smaller than DOFREQ, ACOMEVAL exits.

(d) If the new range uncertainty estimate is smaller than the old one, a counter is incremented (initially set to 0). If the counter reads less than 3 then MEASURE is entered again to try and improve the estimate, otherwise ACOMEVAL exits.

The automatic focussing module is activated by the following messge:

AUTOFOC(EST_RANGE,RANGINVAL,NCUT)

Where: EST_RANG-is the initial range estimate (in inches); RANGINVAL-is the initial search interval (in inches); NCUT-is the value for cut if it is positive, or a flag for automatic setting of CUT if it is negative.

The outputs of the module are the following globals: FOCRANG- is the best range estimate; FOCDELF-is the best range uncertainty estimate; FOCFLG-indicates success or failure of the module and the source of failure.

The resolution needed of the focus servo is approximately the resolution of the scanning circuits (i.e the dimension of a raster cell) and is independent of lens focal length and the range as the following analysis shows:

Take two points at distances X1 and X2 from the lens center. Denote the distances between lens center and vidicon needed to focus on X1 and X2, Y1 and Y2 respectively. Using the focus equation we have approximately:

Y1-Y2=(F*2)*(X2-X1)/(R*2)

where F is the lens focal length and R=(X1+X2)/2. If we want 10 points in a search interval of at least 3*DOFREQ then the increments of foc between two consecutive points will be:

DY=(F*2)*3*DOFREQ/[10*(R*2)]

but:

DOFREQ=(R*2)*F#*C/(F*2)

where C is the scanning resolution. Hence:

DY=3*F#*C/10

Modifications to improve the performance of the module might be of two types:

(a) Modification of the score function computed at each range. One possibility is to try to use the concept of energy of high spatial frequencies used by B. Horn at M.I.T for similar applications [11].

(b) Modification of the search algorithm. Here we use the three highest scores out of ten computed at equidistant points in the search interval, to decide whether to stop the search or what interval to search next. If the measurements were not noisy we could use a kind of binary search which is faster. The problem of finding a maximum of a function from noisy mesurements of its values belongs to the field of "stochastic approximations". Most of the theoretical results of this field are concerned with convergence properties of the algorithms after a large number of measurements and not with their efficiency [14].

BIBLIOGRAPHY

In the following list of references we will use the following abbreviations:

(a) IJCAI-2 for: Proceedings of the 2nd. International Joint Conference on Artificial Intelligence; London; September 1971.

(b) AIM xxx for: Stanford Artificial Intelligence Project Memo no. xxx, Stanford University.

1 Agin, G.J., Representation and description of curved objects, Ph.D Thesis, Artificial Intelligence Project, Stanford University, 1972.

2 Aida, S., et. al., Pattern recognition by artificial taotile sense, IJCAI-2.

3 Barrow, H.G., and Slater, S.H., Design of low-cost equipment for cognitive robot research, In D. Michie and B. Meltzer eds., Machine Intelligence 5, Edinburgh University Press 1969.

4 Brice, C.R., and Fenema, C.L., Scene analysis using regions, Artificial Intelligence Group Technical Note no.17, Stanford Research Insitute.

5 Cornsweet, T.N., Visual perception (book), Academic Press 1970, pp 137 and 256.

6 Ejiri, M., at. al., An intelligent robot with cognition and decision making ability, IJCAI-2.

7 Falk, G., Scene analysis based on imperfect edge data, IJCAI-2.

8 Falk, G., Computer interpretation of imperfect line data as a three dimensional scene, AIM 139, 1970.

9 Feldman, J.A., and Sproull, R.F., System support for the Stanford Hand-Eye system, IJCAI-2.

10 Feldman, J.A., et. al., The use of vision and manipulation to solve the "Instant Insanity" puzzle, IJCAI-2.

11 Horn, B., Focussing, M.I.T. Project MAC, Artificial Intelligence group memo no. 160, May 1968.

12 Hueckel, M.H., An operator which locates edges in digitized pictures, JACM, vol.18, no.1, January 1971.

13 Joutaki, M., et. al., ETL Robot-1: Artificial intelligent robot, Japan Electronic Engineering, Feb. 1971.

14 Loginov, M.V., Methods of stochastic approximations (a survey), Automatuon & Remote Control, Vol. 27, No. 4, 1966.

15 Munson, J.H., Robot planning, execution and monitoring in an uncertain environment, IJCAI-2.

16 Nilsson, N.J., A mobile automaton, an application of artificial intelligence techniques, Proceedings of the 1st International Joint Conference on Artificial Intelligence, Washington D.C., May 1969.

17 Paul, R., Trajectory control of a computer arm, IJCAI-2.

18 Paul, R., Modelling, trajectory calculation and servoing of a computer controlled arm, Ph.D Thesis, Computer Science Dept., Stanford University, Sept. 1972.

19 Pingle, K.K., and Tenenbaum, J.M., An accomodating edge follower, IJCAI-2.

20 Proceedings of the 2nd.International Symposium On Industrial Robots, May 1972, Chicago Illinois:

21 Quam, L.H., Computer compArison of pictures, AIM 144, May 1971.

22 Roberts, L.G., Machine perception of three dimensional solids, M.I.T. Lincoln Lab. Technical Report no. 315, May 1963.

23 Scheinman, V.D., Design of a computer controlled manipulator, AIM 92, June 1969.

24 Schmidt, R.A, A study of the real-time control of a computer driven vehicle, AIM 149.

25 Shirai, Y., and Inoue, H., Visual feedback of robot in assembly, ETL A.I.R. Group memo no. 1, 1972.

26 Sobel, I., Camera models and machine percertion, AIM

121, 1970.

27 Swinehart, D., and Sproull, R., SAIL", Stanford artificial Intelligence Lab. Operating Note no. 57, Nov. 1969.

28 Tenenbaum, J.M., Accomodation in computer vision, AIM 134, Sept. 1970.

29 Wichman, W.M., Use of optical feedback in the computer control of an arm, AIM 56, Aug. 1967.