UAH RESEARCH REPORT NO. 728

APPLICATIONS OF ARTIFICIAL INTELLIGENCE TO SPACE STATION

General Purpose Intelligent Sensor Interface

Final Report for the 1987-1988 Project Year

Prepared by:

James W. McKee
Johnson Research Center
The University of Alabama in Huntsville
Huntsville, Alabama 35899

Prepared for:

John Wolfsberger
System Software Branch
Information and Electronic Systems Lab

Marshall Space Flight Center
National Aeronautics and
Space Administration
Marshall Space Flight Center, AL 35812

September 1988

TABLE OF CONTENTS

APPLICATIONS OF ARTIFICIAL INTELLIGENCE TO SPACE STATION

Task : General Purpose Intelligent Sensor Interface

Final Report for the 1987-1988 Project Year

1. Purpose

This final report describes the accomplishments of the General Purpose Intelligent Sensor Interface task of the Applications of Artificial Intelligence to Space Station grant for the period from October 1, 1987 through September 30, 1988. Portions of the First Biannual Report that have not been revised will not be included in this report but only referenced.

2. Research Goal

The UAH research goal is to develop an intelligent sensor system that will simplify the design and development of expert systems that use sensors of physical phenomena as a source of data. This research will concentrate on the integration of image processing sensors and voice processing sensors with a computer designed for expert system development. The result of this research will be the design and documentation of a system in which the user will not need to be an expert in such areas as image processing algorithms, local area networks, image processor hardware selection or interfacing, television camera selection, voice recognition hardware selection, or analog signal processing. The user will be able to access data from video or voice sensors through standard LISP statements without any need to know about the sensor hardware or software.

3. Research Approach

This research project is divided into three phases. This report represents the status of the project at the end of the first phase.

3.1 Phase one

The first phase concentrated on interfacing an image processing hardware system with a LISP machine. This phase was divided into the following tasks:

1. Survey and select a LISP machine on which to implement the LISP portion of the project;
2. Survey and select a hardware image processing system on which to implement the image processing functions;
3. Determine a protocol between the two machines;
4. Write a specification defining the functions on the LISP

machine and their corresponding actions on the image processing machine;

5. Write and debug the software on the machines;

6. Integrate the hardware and software and test the system; and

7. Demonstrate the system by having the user do simple image processing operations on the LISP machine.

3.2 Phase two

The second phase of this project will be devoted to determining what voice processing system can be interfaced to the LISP machine and to expand the library of basic image processing functions on the PC. (Because of the design of the interface, these functions will be available on the LISP machine after a calling function is written on the LISP machine.) A survey of voice processing hardware will be conducted. If possible a voice processing hardware system will be assembled. The software will be written and tested to interface the voice system with the LISP machine. The library of basic image processing functions developed in phase one will be expanded to include primitive image analysis functions such as histogram analysis functions, profile (projections) computation function, thresholding functions, edge finding functions, etc.

3.3 Phase three

In the third phase of this project a voice output system will be assembled and interfaced to the LISP machine. Work will be continued on the development of voice recognition software on the LISP machine and in the voice recognition system. Also higher level image analysis functions will be added to the library of image processing functions.

4. Phase one -- video sensor interface

This section reports on the interface of a video system with the LISP machine. The first phase has been divided into five tasks: survey and select a good image processing card, survey and select a good monochrome solid state camera, write the software to interface the image processing card to the PC, write the software to interface the PC to the Symbolics, and on the Symbolics write the software to parse LISP commands and interface to the PC.

A quick survey of the image processing hardware market indicated that image processing hardware can be grouped by price (and performance) into two groups: cards costing less than $10,000 (that plug into ATs or XTs) and cards or systems costing over $20,000 that either stand alone, have an adapter card that plugs into an AT, or plug into other higher speed busses such as the VME bus. All of the later cards or system have a much higher performance (at least an order of magnitude) than the cards in the first group. Since this project does not require the higher

performance, it was decided to concentrate on cards in the first group. The results of a survey of commercial vendor of image processing cards that plug into an AT is shown in Appendix A of the First Biannual Report. As would be expected the price/performance ratio is very close for the group of cards. But it is our opinion that the Matrox MVP-AT card gives the best price/performance at the present time.

Appendix B in the First Biannual Report shows the result of the survey of commercially available solid state monochrome cameras. Color cameras were not surveyed because they are more than twice as expensive and have less than half the resolution of the monochrome cameras. Solid state cameras, as opposed to tube cameras, are used in image processing applications because they have no geometric distortion. Again as expected the price/performance ratio of the cameras was close. But it is our opinion that the Sony XS-77 gives the best price/performance at this time.

Various local area networks were considered for the communication between the PC and the Symbolics. The two major candidates were Kermit on RS-232 and plain RS-232. The Symbolics has Kermit in LISP. The C source code for Kermit was obtained from Columbia University (which has Kermit source in almost every language). (See Appendix C in the First Biannual Report for ordering information.) After evaluating the code and estimating the time to bring Kermit up on the PC, it has been decided to use RS-232 and no higher level protocol for the present time. If the link is prone to error and a more reliable LAN is need, the functions in Kermit can be incorporated into the software on the PC.

Both the TI Explorer and the Symbolics computers were considered as the host for the LISP portion of the image processor development. Both machines are designed to execute LISP. The Symbolics machine was chosen as the test bed because the researchers involved with the LISP portion of the task have had extensive experience using the Symbolics and the Symbolics has available a color monitor which will allow us to display grayscale images captured by the image processing system. Since everything developed on the Symbolics will be written in LISP, the functions should be portable to any machine that executes LISP.

An Matrox MVP-AT image processing board that plugs into a PC AT, a Packard Bell AT, a Sony XC-77 monochrome television camera, and a NEC Multisync color monitor (used for image output from the Matrox card) have been borrowed from other projects and are being used to develop the interface between the PC and the image processing hardware. The image processing code is being written in C on the PC. Figure 4.1 shows a hardware block diagram of the system as it is presently configured.

Appendix A is a copy of the software design specification for the image processing software that will be implemented on the PC and the Symbolics. For each command the specification defines the format of the command on the Symbolics, i.e., the name of the command and the list structure for any data to be passed to the PC, and to what the function evaluates (the data returned from the PC). The specification defines the protocol between the LISP machine and the PC. The specification also defines what image processing function the PC will perform in response to the commands for the LISP machine.

Appendix B is a listing of the LISP programs developed on the Symbolics. These functions should be directly portable to any LISP machine. Included in the listing are the image processing functions, the parsing functions and the communication functions other than the functions standard in the operating system.

Appendix C is the copy of the structured control flow diagrams for the software that is being written on the PC. These control flow diagrams show what actions will be performed when each image processing function is executed on the PC.

Appendix D is a copy of the source code listings for the image processing functions and communication that run on the PC. These listing show in detail how each image processing function on the Symbolics machine is executed on the PC.

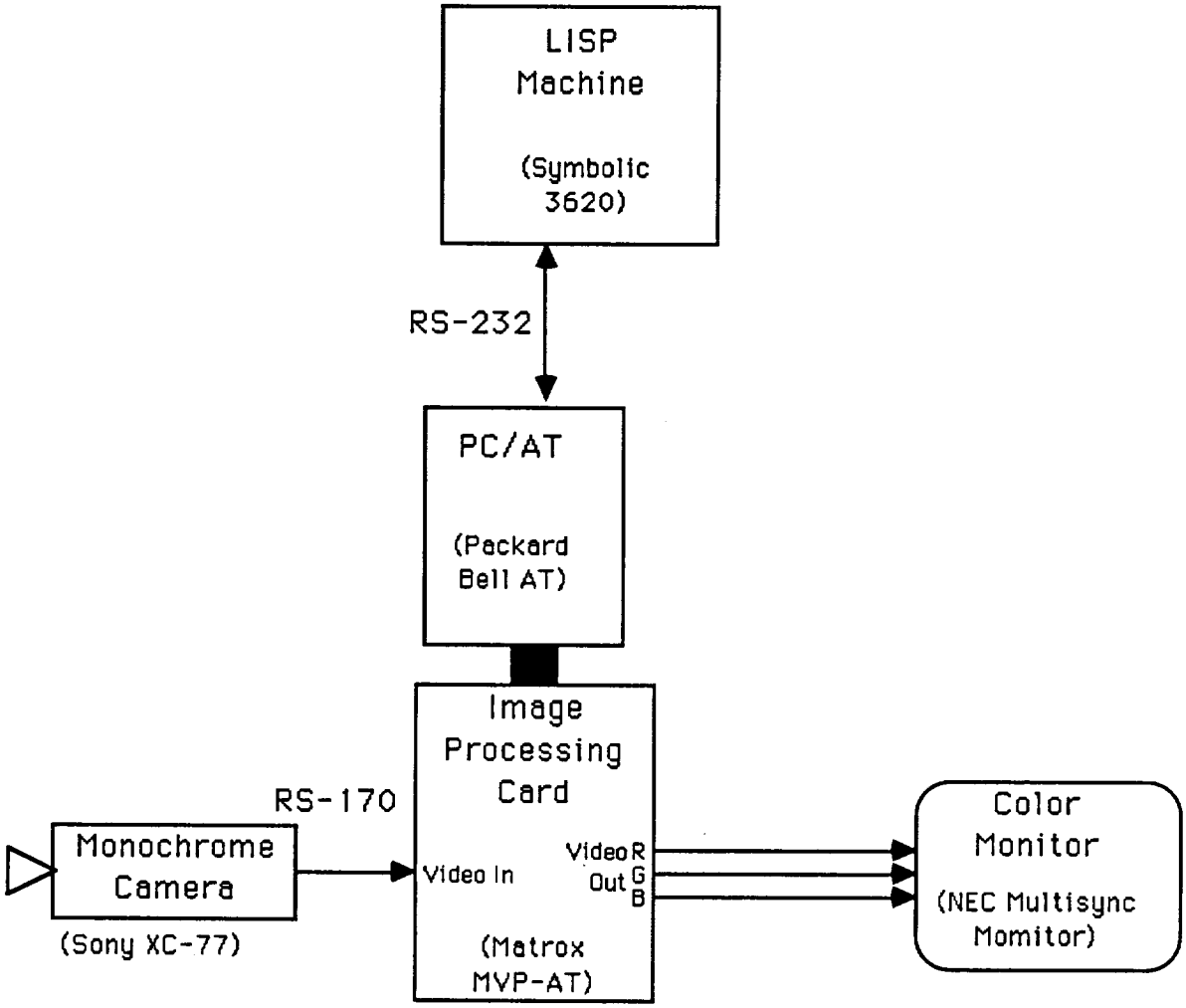


Figure 4.1 Hardware Block Diagram

Appendix A Symbolic Image Processing Software Specification

This specification defines the user interface to the symbolic image processing software, the presentation level software in a LISP machine, the interface between the LISP machine and the image processing machine, and the operations of the image processor.

User Interface

All symbolic image processing functions are of the form

(image-function-name argument-list)

where image-function-name it the name of the image processing function to be performed and argument-list evaluates to a list of parameters for the image processing operation. The function will evaluate to **nil** if there was an error in executing the image processing function on the PC, and an error statement will be displayed on the monitor. Any other non-**nil** value is the returned data. The form of the data is function dependent. If no data is returned, the function will evaluate to **T**. If only one number is returned, the function will evaluate to the number. If more than one number is returned, the function will evaluate to a list of the numbers or an array (function dependent).

LISP Machine -- Image processing Machine Interface

All communication between the LISP machine and the image processing machine will be in the form of ASCII files consisting of strings of ASCII printable characters separated by blanks; the end of the file will be the character string "END" followed by at least one blank character.

The control will be in the form of master-slave. The LISP machine will be the master and will send commands to the image processor machine. The image processor machine will respond with a series of two messages to each command from the LISP machine. The first message will start with the character string "0" (zero) if the command can be executed or an error code (a character string other than the character string "1" (one)) if the command can not be executed for any reason. The second message will start with the character string "1" (one) if the command was successfully executed and then data, if any, in a predefined format dependent on the particular command. If the command was not executed successfully, then an error code string will be returned.

The first string in the file sent by the LISP machine will be the predefined name of the image processing command. Following the

first string will be a predetermined number (function dependent) of strings of data for the image processing function.

The following list defines the image processing function and the structure of the parameter list. It also defines the exact mnemonics and data structure for the series of commands from the LISP machine and the responses by the image processing machine to the commands, where (number) means the previous item repeated "number" times.

List of Image Processing Commands

General Image Functions

```
initialize                    ()
     evaluates to:    T

command: [INIT END]
response:[0 END]
response:[1 END]
```

Initialize the image processor hardware system; must be done before any image processing functions are invoked.

```
get-number-frames             ()
     evaluates to:    number

command: [#FRM number END]
response:[0 END]
response:[1 number END]
```

Returns in "number" the number of frames in the system.

```
get-frame-size                ()
     evaluates to:    (frame-x-size frame-y-size)

command: [FRM-SZ END]
response:[0 END]
response:[1 frame-x-size frame-y-size END]
```

Returns the x and y dimension of the frames in the system.

```
quit                          ()
     evaluates to:    T

command: [QUIT END]
response:[0 END]
response:[1 END]
```

Causes the program running on the PC to exit. Returns control of the PC to DOS.

```
select-camera                    (number)
     evaluates to:   T

command: [#CAM number END]
response:[0 END]
response:[1 END]
```

Sets the selected or default camera input to port "number". This camera will be used for all the image acquisitions operations until another camera is selected.

```
live                             ()
     evaluates to:   T

command: [LIVE END]
response:[0 END]
response:[1 END]
```

Puts the image processor in the live mode. The image seen by the selected camera will be displayed on the monitor.

```
snap                        (fb-1 )
     evaluates to:   T

command: [SNAP fp-1 END]
response:[0 END]
response:[1 END]
```

Store an image from the selected camera into the "fb-1" frame buffer.

```
display-on                  (fb-1)
     evaluates to:   T

command: [ON fb-1 END]
response:[0 END]
response:[1 END]
```

Routes the image in "fb-1" to the monitor.

```
display-off                ()
     evaluates to:    T

command: [OFF END]
response:[0 END]
response:[1 END]
```

Turns the display on the monitor off.

```
get-number-process-map    ()
     evaluates to:   (process-map-number  number-of-process-
maps)

command: [#P-MAP END]
response:[0 END]
response:[1 process-map-number number-of-process-maps END]
```

Returns the number of the selected process map and the total number of process maps available to the user. Process map numbers range from 1 to number-of-process-maps.

```
select-process-map              (number)
     evaluates to:   T

command: [P-MAP# number END]
response:[0 END]
response:[1 END]
```

Sets process map "number" as the active or default process map. All process map operations will use this process map.

```
read-process-map    (starting-index number-of-cells)
     evaluates to:   an array

command: [R-P-MAP starting-index number-of-cells END]
response:[0 END]
response:[1 data (number-of-cells) END]
```

Transfers "number-of-cells" cells to an array from the selected process map starting at "starting-index" of the process map.

```
write-process-map   (array-name starting-index number-of-cells)
     evaluates to:   T

command: [W-P-MAP starting-index number-of-cells data (number-of-
cells) END]
response:[0 END]
response:[1 END]
```

Transfers "number-of-cells" cells from the array "array-name" to the selected process map starting at "starting-index" of the process map.

```
get-number-output-maps   ()
     evaluates to:   (output-map-number number-of-output-maps)

command: [#O-MAP END]
response:[0 END]
response:[1 output-map-number number-of-output-maps END]
```

Returns the number of the selected output map and the number of available output maps. Output map numbers range from 1 to number-of-output-maps.

```
select-output-map                (number)
     evaluates to:   T

command: [O-MAP# number END]
response:[0 END]
response:[1 END]
```

Sets output map "number" as the active or default output map. All output map operations will use this output map.

```
select-color-map                 (number)
     evaluates to:   T

command: [C-MAP# number END]
response:[0 END]
response:[1 END]
```

Sets color map "number" as the active or default color map. All color map operations will use this color map. Red = 0; Green = 1; Blue = 2.

```
read-output-map     (starting-index number-of-cells)
     evaluates to:   an array

command: [R-O-MAP starting-index number-of-cells END]
response:[0 END]
response:[1 data {number-of-cells} END]
```

Transfers "number-of-cells" cells to an array from the selected color map of the selected output map starting at "starting-index" of the color map.

```
write-output-map    (array-name starting-index number-of-cells)
     evaluates to:   T

command: [W-O-MAP starting-index number-of-cells data (number-of-
cells) END]
response:[0 END]
response:[1 END]
```

Transfers "number-of-cells" cells from the array "array-name" to the selected color map of the selected output map starting at "starting-index" of the color map.

```
process-image          (fb-1 window-1 fb-2 window-2)
     evaluates to:    T

command: [MOV fb-1 ulx-1 uly-1 lrx-1 lry-1 fb-2 ulx-2 uly-2 lrx-2
lry-2 END]
response:[0 END]
response:[1 END]
```

Pass the portion of the image starting in the upper left hand corner of window-1 of fb-1 through the selected process map and store the resultant image in window-2 of fb-2. The size of the image is that of window-2; window-1 is used only as the starting position for the data.

```
transfer-image              (fb-1 window-1)
     evaluates to:    an array

command: [TRANS fb ulx uly lrx lry END]
response:[0 END]
response:[1 (lrx - ulx) (lry - uly) block-of-image-data END]
```

Transfer the portions of the image in window-1 of fb-1 to the LISP machine placed in an array. The data will be sent in three blocks of ASCII characters: the first string will be the x dimension of the block, the second string will be the y dimension of the block, and the third string will be the intensity (pixel) information in the window. Each pixel will be divided into two nibels and coded as hex characters, most significant nibel first. There will be 2 * number-of-pixels per row * number-of-rows characters in the third block.

```
copy                    (fb-1 window-1 fb-2 window-2)
     evaluates to:    T

command: [COPY fb-1 ulx-1 uly-1 lrx-1 lry-1 fb-2 ulx-2 uly-2
lrx-2 lry-2 END]
response:[0 END]
response:[1 END]
```

Copy a portion of an image in fb-1 to the window-2 of fb-2. The window size of window-2 is used as the size of window-1 in fb-1. The upper left hand corner of window-1 is used as the starting point for the window in fb-1.

```
subtract                (fb-1 window-1 fb-2 window-2 fb-3 window-3)
     evaluates to:   T

command: [SUB fb-1 ulx-1 uly-1 lrx-1 lry-1 fb-2 ulx-2 uly-2 lrx-2
lry-2 fb-3 ulx-3 uly-3 lrx-3 lry-3 END]
response:[0 END]
response:[1 END]
```

Subtract portions of an image in fb-2 from an image in fb-1 and place the difference in window-3 of fb-3. The window size of window-3 is used as the size of windows in fb-1 and fb-2. The upper left hand corner of window-1 and window-2 are used as the starting point for the windows in their respective frames.

```
add                     (fb-1 window-1 fb-2 window-2 fb-3 window-3)
     evaluates to:   T

command: [ADD fb-1 ulx-1 uly-1 lrx-1 lry-1 fb-2 ulx-2 uly-2 lrx-2
lry-2 fb-3 ulx-3 uly-3 lrx-3 lry-3 END]
response:[0 END]
response:[1 END]
```

Add portions of images in fb-1 and fb-2 together and place the sum in window-3 of fb-3. The window size of window-3 is used as the size of windows in fb-1 and fb-2. The upper left hand corner of window-1 and window-2 are used as the starting point for the windows in their respective frames.

```
multiply                (fb-1 window-1 fb-2 window-2 fb-3 window-3)
     evaluates to:   T

command: [MULT fb-1 ulx-1 uly-1 lrx-1 lry-1 fb-2 ulx-2 uly-2
lrx-2 lry-2 fb-3 ulx-3 uly-3 lrx-3 lry-3 END]
response:[0 END]
response:[1 END]
```

Multiply portions of images in fb-1 and fb-2 together and place the product in window-3 of fb-3. The window size of window-3 is used as the size of windows in fb-1 and fb-2. The upper left hand corner of window-1 and window-2 are used as the starting point for the windows in their respective frames.

```
get-number-coef-banks          ()
     evaluates to:   (number-of-banks dimension)

command: [#COEF END]
response:[0 END]
response:[1 number-of-banks dimension END]
```

Returns the number of coefficient banks and their dimension.

```
select-coef-bank          (number)
     evaluates to:   T

command: [COEF# number END]
response:[0 END]
response:[1 END]
```

Selects the active or default coefficient bank. All convolutions will use the coefficient information from the "number" coefficient bank. Also all coefficient reads and writes will use "number" coefficient bank.

```
read-coefficients   (starting-index number-of-cells)
     evaluates to:   an array

command: [R-COEF starting-index number-of-cells END]
response:[0 END]
response:[1 data (number-of-cells) END]
```

Transfers "number-of-cells" cells to an array from the selected coefficient array starting at "starting-index" of the coefficient array.

```
write-coefficients  (array-name starting-index number-of-cells)
     evaluates to:   T

command: [W-COEF starting-index number-of-cells data (number-of-
cells) END]
response:[0 END]
response:[1 END]
```

Transfers "number-of-cells" cells from the array "array-name" to the selected coefficient array starting at "starting-index" of the coefficient array.

```
convolve              (fb-1 window-1 fb-2 window-2 number-of-rows
number-of-columns)
     evaluates to:    T

command: [CONV fb-1 ulx-1 uly-1 lrx-1 lry-1 fb-2 ulx-2 uly-2
lrx-2 lry-2 number-of-rows number-of-columns END]
response:[0 END]
response:[1 END]
```

Convolve a portion of an image in frame fb-1 inside the window defined by window-2 using a the upper-left hand corner of window-1 as the starting point. The convolution uses the coefficients in the selected (default) coefficient-array; the size of the convolution kernel is defined by number-of-rows and number-of-columns. The user is responsible for insuring that valid kernel data is in the selected coefficient bank before calling the convolution function. The results of the convolution are placed in window-2 of fb-2.

```
read-values          (starting-index number-of-cells)
     evaluates to:    an array

command: [R-VAL start-address number-of-values  END]
response:[0 END]
response:[1 data (number-of-values) END]
```

Transfers "number-of-cells" cells to an array from the histogram-profile array starting at "starting-index" of the histogram-profile array.

```
histogram            (fb-1 window-1)
     evaluates to:    T

command: [HIST fb-1 ulx uly lrx lry END]
response:[0 END]
response:[1 END]
```

Do a histogram on a portion of the image within window-1 of fb-1. The results are stored in the histogram-profile array.

```
profile-x            (fb-1 window-1)
     evaluates to:    T

command: [PRO-X fb-1 ulx uly lrx lry END]
response:[0 END]
response:[1 END]
```

Do a profile in the x-direction on a portions of the image within window-1 of fb-1. The results are stored in the histogram-profile array.

```
profile-y            (fb-1 window-1)
     evaluates to:    T

command: [PRO-Y fb-1 ulx uly lrx lry END]
response:[0 END]
response:[1 END]
```

Do a profile in the y-direction on a portions of the image within window-1 of fb-1. The results are stored in the histogram-profile array.

Definition of Variables

fb-1 -- frame buffer number 1; 1 <= fb-1 <= number of frame buffers

fb-2 -- frame buffer number 2; 1 <= fb-2 <= number of frame buffers

fb-3 -- frame buffer number 3; 1 <= fb-3 <= number of frame buffers

max-x -- the number of pixels in a row of the frame buffer

max-y -- number of rows of pixels in a frame buffer

window-1 -- (ulx uly lrx lry) list of the coordinates of the window
- ulx -- upper left x coordinate; 0 <= ulx <= max-x - 1.
- uly -- upper left y coordinate; 0 <= uly <= max-y - 1.
- lrx -- lower right x coordinate; 1 <= lrx <= max-x.
- lry -- lower right y coordinate; 1 <= lrx <= max-y.

window-2 -- (ulx uly lrx lry) list of the coordinates of the window
- ulx -- upper left x coordinate; 0 <= ulx <= max-x - 1.
- uly -- upper left y coordinate; 0 <= uly <= max-y - 1.
- lrx -- lower right x coordinate; 1 <= lrx <= max-x.
- lry -- lower right y coordinate; 1 <= lrx <= max-y.

window-3 -- (ulx uly lrx lry) list of the coordinates of the window
- ulx -- upper left x coordinate; 0 <= ulx <= max-x - 1.
- uly -- upper left y coordinate; 0 <= uly <= max-y - 1.
- lrx -- lower right x coordinate; 1 <= lrx <= max-x.
- lry -- lower right y coordinate; 1 <= lrx <= max-y.

histogram-profile array -- long hist-prof[512];
An array of 512 long numbers which can be used to store the results of a histogram or profile command; warning: will be written over by the next histogram or profile command.

coefficient-array -- short coef[system-defined][at-least-50]
Arrays, of at least 50 short numbers each, are available to store the coefficients used in convolutions function. The number are stored row by row. There should be enough storage for up to a 7 by 7 convolution kernel (7 * 7).

Appendix B Listing of LISP code

B.1 List of LISP function calls

```
;;; -*-     Syntax: Common-Lisp;
;;;         Package: USER;
;;;         Base: 10;
;;;         Mode: LISP    -*-


(defun test ()
  (issue-command "TEST" :return-as-image-array t))

(defun initialize ()
  (issue-command "INIT")
  t)

(defun get-number-frames ()
  (issue-command "#FRM"))

(defun get-frame-size ()
  (issue-command "FRM-SZ"))

(defun select-camera (n)
  (issue-command "#CAM"
           :outgoing-arguments
           (list n))
  t)

(defun live ()
  (issue-command "LIVE")
  t)

(defun snap (fb-1)
  (issue-command "SNAP"
           :outgoing-arguments
           (list fb-1))
  t)

(defun display-on (fb-1)
  (issue-command "ON"
           :outgoing-arguments
           (list fb-1))
  t)

(defun display-off ()
  (issue-command "OFF")
  t)

(defun get-number-process-map ()
  (issue-command "#P-MAP"))
```

```
(defun select-process-map (n)
  (issue-command "P-MAP#"
           :outgoing-arguments
           (list n))
  t)

(defun read-process-map (starting-index number-of-cells)
  (issue-command "R-P-MAP"
           :outgoing-arguments
           (list starting-index number-of-cells)))

(defun write-process-map (array-name starting-index
                  number-of-cells)
  (issue-command "W-P-MAP"
           :outgoing-arguments
           (append (list  starting-index
                    number-of-cells)
                (loop for element from
                       starting-index
                       to (- (+ starting-index
                             number-of-cells)
                          1)
                     collect (aref array-name
                             element))))
  t)


(defun get-number-output-maps ()
  (issue-command "#O-MAP"))

(defun select-output-map (n)
  (issue-command "O-MAP#"
           :outgoing-arguments
           (list n))
  t)

(defun select-color-map (n)
  (issue-command "C-MAP#"
           :outgoing-arguments
           (list n))
  t)

(defun read-output-map (starting-index number-of-cells)
  (issue-command "R-O-MAP"
           :outgoing-arguments
           (list starting-index number-of-cells)
           :return-arguments-in-array
           number-of-cells))

(defun write-output-map (array-name starting-index
                 number-of-cells)
```

```
  (issue-command "W-O-MAP"
           :outgoing-arguments
           (append (list  starting-index
                    number-of-cells)
                (loop for element
                      from starting-index
                      to (- (+ starting-index
                               number-of-cells)
                            1)
                     collect (aref array-name
                                   element))))
  t)

(defun process-image (fb-1 window-1 fb-2 window-2)
  (issue-command "MOV"
           :outgoing-arguments
           (append (list fb-1) window-1
                (list fb-2) window-2))
  t)

(defun transfer-image (fb-1 window-1)
  (issue-command "TRANS"
           :outgoing-arguments
           (append (list fb-1) window-1)
           :return-as-image-array t))

(defun copy (fb-1 window-1 fb-2 window-2)
  (issue-command "COPY"
           :outgoing-arguments
           (append (list fb-1) window-1
                (list fb-2) window-2))
  t)

(defun subtract (fb-1 window-1 fb-2 window-2 fb-3 window-3)
  (issue-command "SUB"
           :outgoing-arguments
           (append (list fb-1) window-1
                (list fb-2) window-2
                (list fb-3) window-3))
  t)

(defun add (fb-1 window-1 fb-2 window-2 fb-3 window-3)
  (issue-command "ADD"
           :outgoing-arguments
           (append (list fb-1) window-1
                (list fb-2) window-2
                (list fb-3) window-3))
  t)
```

```
(defun multiply  (fb-1 window-1 fb-2 window-2 fb-3 window-3)
  (issue-command "MULT"
           :outgoing-arguments
           (append (list fb-1) window-1
                (list fb-2) window-2
                (list fb-3) window-3))
  t)

(defun get-number-coef-banks ()
  (issue-command "#COEF" ))

(defun select-coef-bank (n xsize ysize)
  (issue-command "COEF#"
           :outgoing-arguments
           (list n xsize ysize))
  t)

(defun write-coefficients (array-name starting-index
                   number-of-cells)
  (issue-command "W-COEF"
           :outgoing-arguments
           (append (list starting-index
                      number-of-cells)
                (loop for element
                       from starting-index
                       to (- (+ starting-index
                             number-of-cells)
                          1)
                      collect (aref array-name
                                element))))
  t)

(defun read-coefficients (starting-index number-of-cells)
  (issue-command "R-COEF"
           :outgoing-arguments
           (list starting-index number-of-cells)
           :return-arguments-in-array
           number-of-cells))

(defun convolve (fb-1 window-1 fb-2 window-2)
  (issue-command "CONV"
           :outgoing-arguments
           (append (list fb-1) window-1
                (list fb-2) window-2))
  t)
```

```
(defun read-values (starting-index number-of-cells)
  (issue-command "R-VAL"
           :outgoing-arguments
           (list starting-index number-of-cells)
           :return-arguments-in-array
           number-of-cells))

(defun histogram (fb-1 window-1)
  (issue-command "HIST"
           :outgoing-arguments
           (append (list fb-1) window-1))
  t)

(defun profile-x (fb-1 window-1)
  (issue-command "PRO-X" :outgoing-arguments
           (append (list fb-1) window-1))
  t)

(defun profile-y (fb-1 window-1)
  (issue-command "PRO-Y"
           :outgoing-arguments
           (append (list fb-1) window-1))
  t)
```

B.2 List of LISP support functions

```
;;; -*-     Syntax: Common-lisp;
;;;         Package: USER;
;;;         Base: 10; Mode: LISP -*-


(defvar serial-stream (si:make-serial-stream
                :baud 9600
                :parity nil
                :number-of-data-bits 8
                :number-of-stop-bits 1
                :xon-xoff-protocol t
                :generate-xon-xoff t
                :request-to-send t
                :ascii-characters t
                :unit 0))

(defun fact (n)
  (cond ((zerop n) 1)
     (t (* n (fact (- n 1))))))

(defvar test-array (make-array '(80 80) :initial-element 0))

(defun test ()
  (send serial-stream :clear-input)
  (loop for i from 0 to 79
     do
    (loop for j from 0 to 79
       do
      (fact 5)
      (zl:aset (send serial-stream :tyi) test-array i j))))
    (send-atom (ascii-to-char 19))))

(defun show-array ()
  (format t "~2%")
  (loop for i from 0 to 79
     do
    (loop for j from 0 to 79
       do
      (format t "~A" (aref test-array i j)))))

(defun send-atom (atom)
  (format serial-stream "~A " atom)
  (send serial-stream :force-output))
```

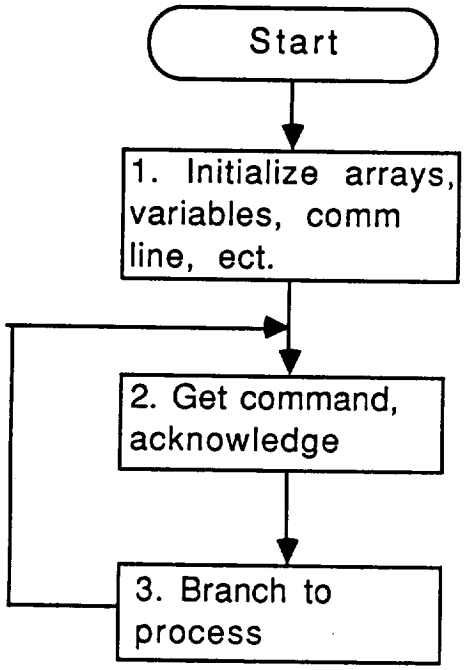
```
(defun issue-command (mnemonic
                      &key (outgoing-arguments nil)
                      (return-arguments-in-array nil)
                      (return-as-image-array nil))
  (setq  chars-read 0)
  (send serial-stream :clear-input)
  (send-atom mnemonic)
  (loop for each-argument in outgoing-arguments
     do
        (send-atom each-argument))
  (send-atom 'end)
  (let ((good-or-bad (recieve-atom))
     (next-atom))
    (cond ((not (zerop good-or-bad))
        (handle-error good-or-bad))
       ((not (equal (setq next-atom (recieve-atom))
                    'end))
        (handle-error 'improper-eof next-atom))
       (t (accept-results return-arguments-in-array
                    return-as-image-array)))))

(defun accept-results (array-flag image-flag)
  (let ((good-or-bad (recieve-atom))(array)(size))
    (cond ((= good-or-bad 1)
        (cond (image-flag
            ;;If result is to be an image array.
            (setq size (list (recieve-atom)
                             (recieve-atom)))
            (setq array (make-array size))
            (loop for row from 0 to (- (car size) 1)
               do
              (loop for col from 0
                       to (- (cadr size) 1)
                 do
                (format t "[~A ~A]" row col)
                (zl:aset (hex-chars-to-integer
                     (get-char serial-stream)
                     (get-char serial-stream))
                     array row col)))
           array)
          (array-flag
           ;;If result is to be an array.
           (setq array (make-array
                    (list array-flag)))
           (loop for col from 0 to (- array-flag 1)
              do
             (zl:aset (recieve-atom) array col))
           array)
          (t
           ;;If result is to be a list.
           (loop as atom = (recieve-atom)
              until (equal atom 'end)
```

```
                collect atom))))
          (t (handle-error good-or-bad)))))

(defun handle-error (code &rest error-info)
  (format t "~2%There was an error, code: ~A" code)
  (cond (error-info
      (format t "    Extra error info: ~A" error-info)))
  code)

(defun recieve-atom ()
  (loop for char = (return-first-non-space) then
        (get-char serial-stream)
     until (char-equal char #\space)
     for incoming-atom = char then
        (string-append incoming-atom char)
     finally (return
           (zl:read-from-string
             (format nil "~A" incoming-atom)))))

(defun return-first-non-space ()
  (loop as char = (get-char serial-stream)
        until (not (char-equal char #\space))
     finally (return char)))


(defvar chars-read)

(defun get-char (stream)
  (setq  chars-read (+ chars-read 1))
  (cond ((=  chars-read 300)
      (loop until (send stream :listen)
            do
        (format stream "%")
        (send stream :force-output))
      (format t "~%I got a [~A]" (send stream :tyi))
      (setq chars-read 0)))
  (send stream :tyi))

(defun hex-chars-to-integer (char1 char2)
  (+ (* (hex-char-to-integer char1) 16)
     (hex-char-to-integer char2)))
```

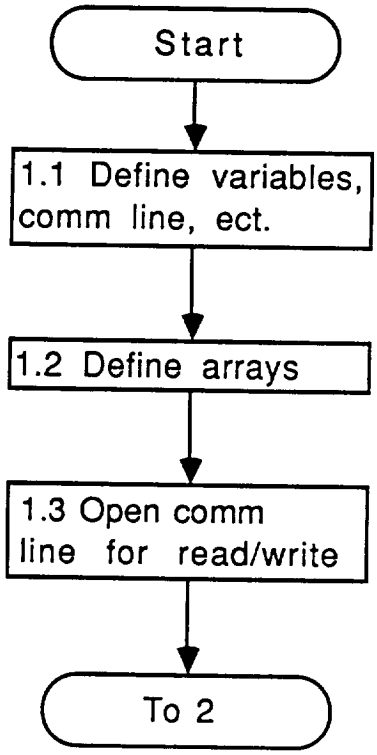
```
(defun hex-char-to-integer (char)
  (case char
    (#\0 0)
    (#\1 1)
    (#\2 2)
    (#\3 3)
    (#\4 4)
    (#\5 5)
    (#\6 6)
    (#\7 7)
    (#\8 8)
    (#\9 9)
    (#\a 10)
    (#\b 11)
    (#\c 12)
    (#\d 13)
    (#\e 14)
    (#\f 15)))
```

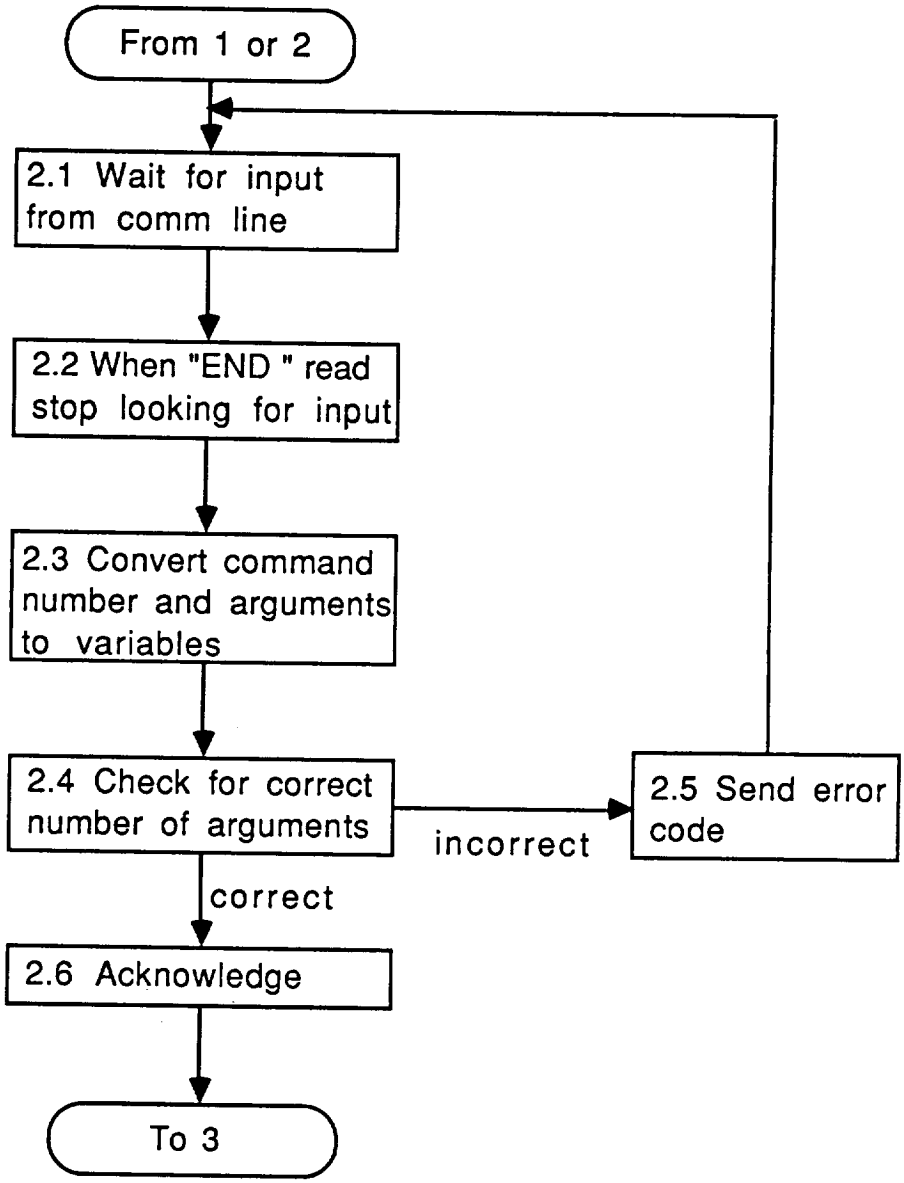
Appendix C Structured Control Flow Diagrams for PC Software

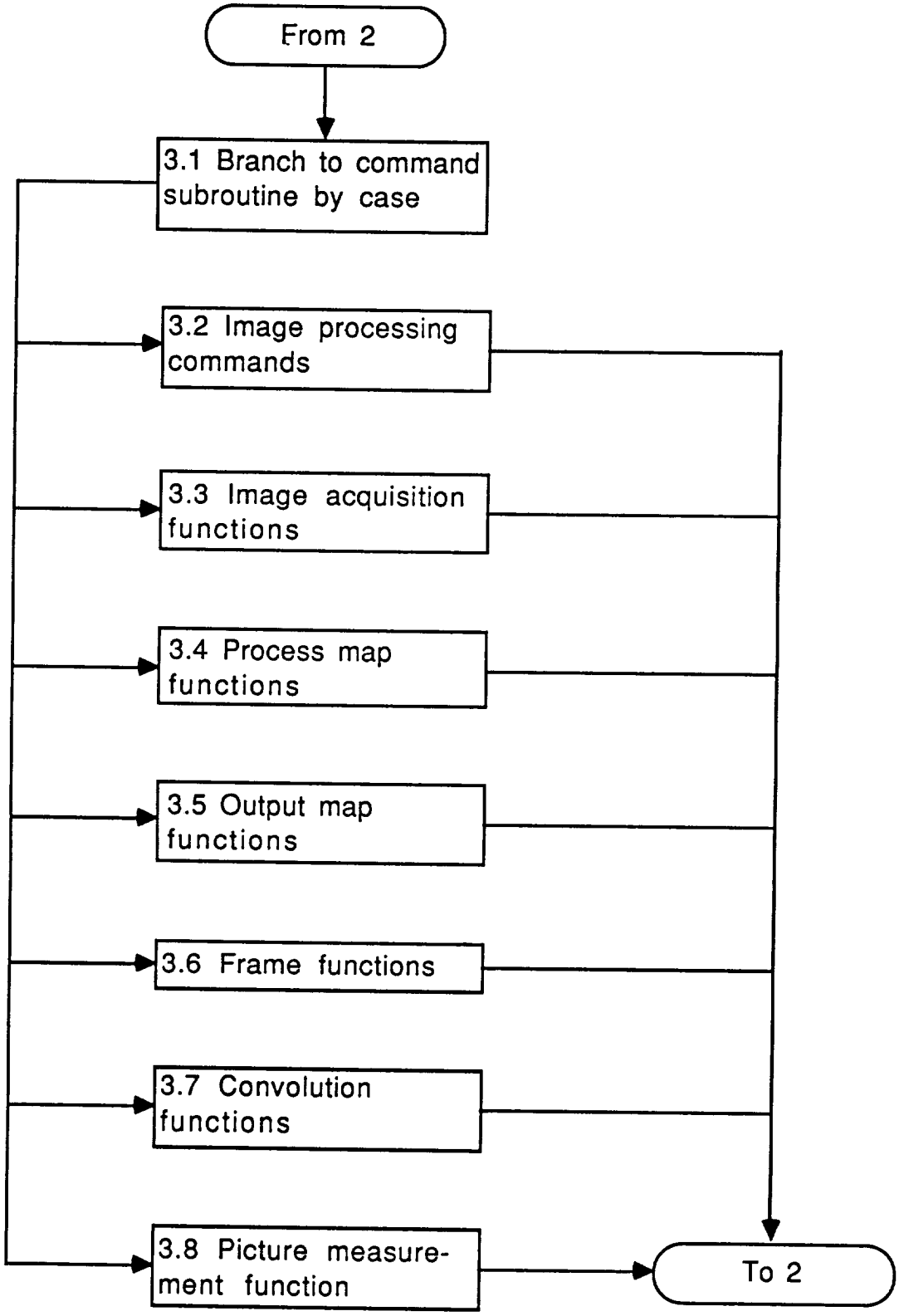
Sheet 0


Start
1. Initialize arrays, variables, comm line, ect.
2. Get command, acknowledge
3. Branch to process

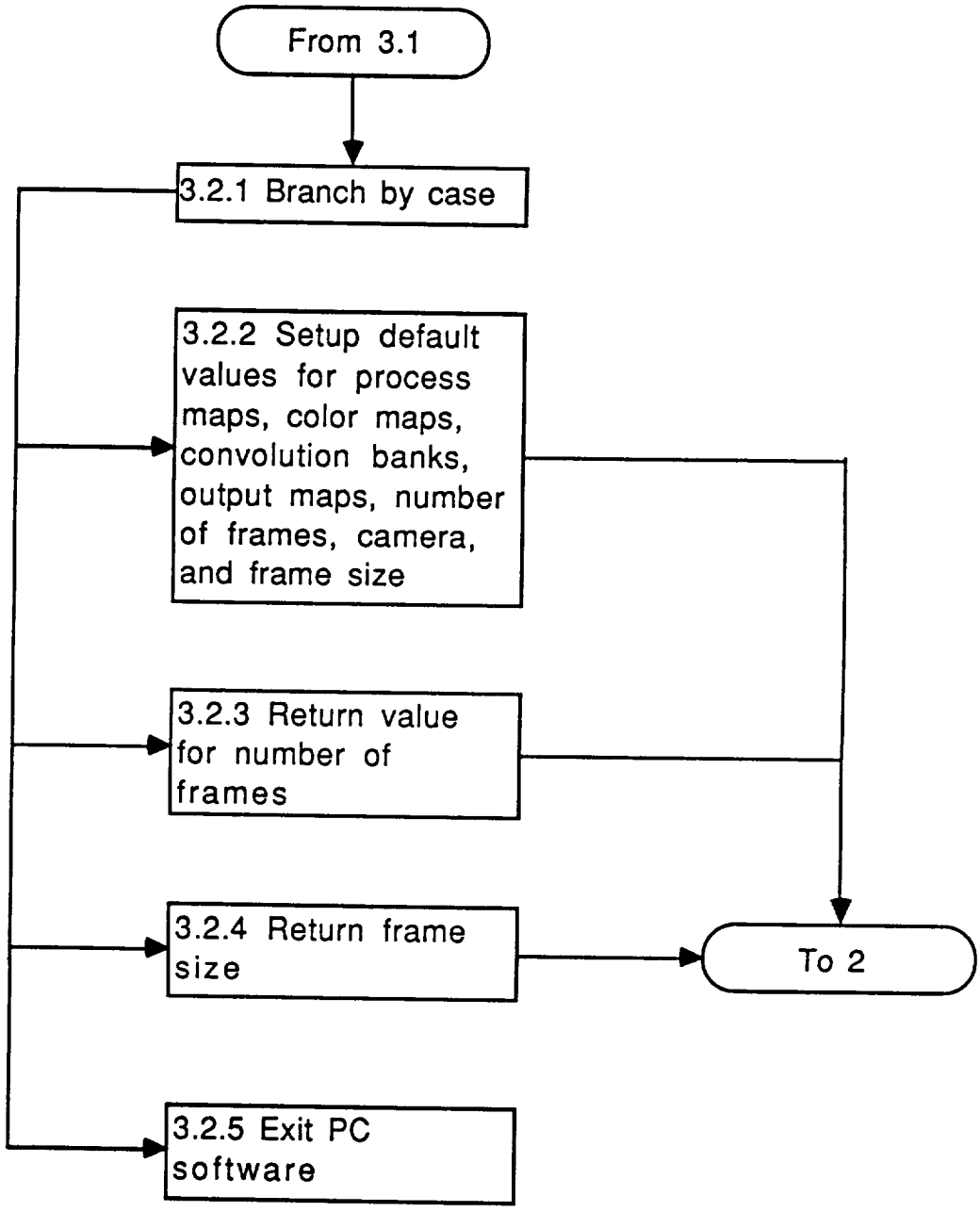
Sheet 1


Start
1.1 Define variables, comm line, ect.
1.2 Define arrays
1.3 Open comm line for read/write
To 2

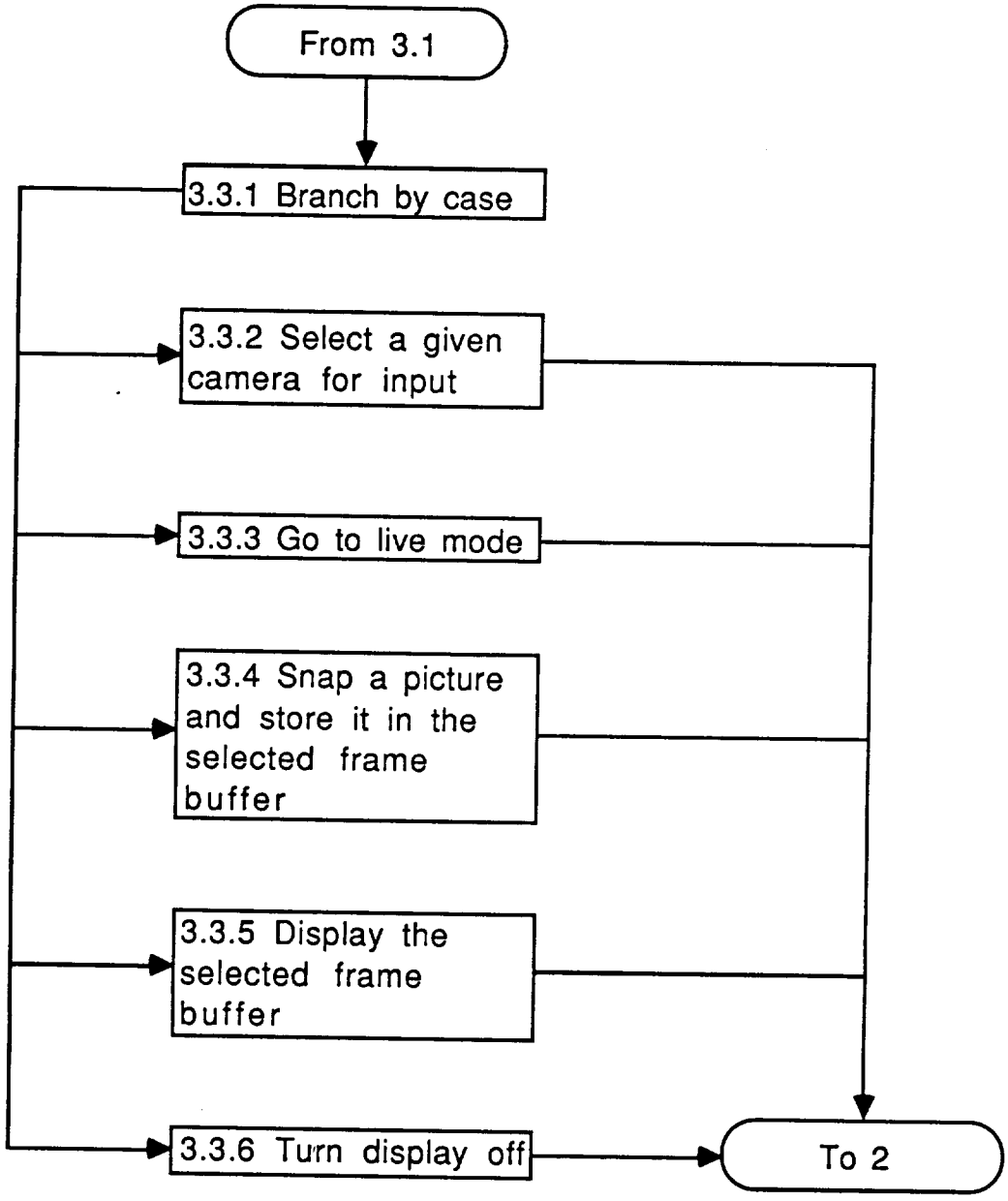

From 1 or 2
2.1 Wait for input from comm line
2.2 When "END " read stop looking for input
2.3 Convert command number and arguments to variables
2.4 Check for correct number of arguments
incorrect
2.5 Send error code
correct
2.6 Acknowledge
To 3


From 2
3.1 Branch to command subroutine by case
3.2 Image processing commands
3.3 Image acquisition functions
3.4 Process map functions
3.5 Output map functions
3.6 Frame functions
3.7 Convolution functions
3.8 Picture measure-ment function
To 2

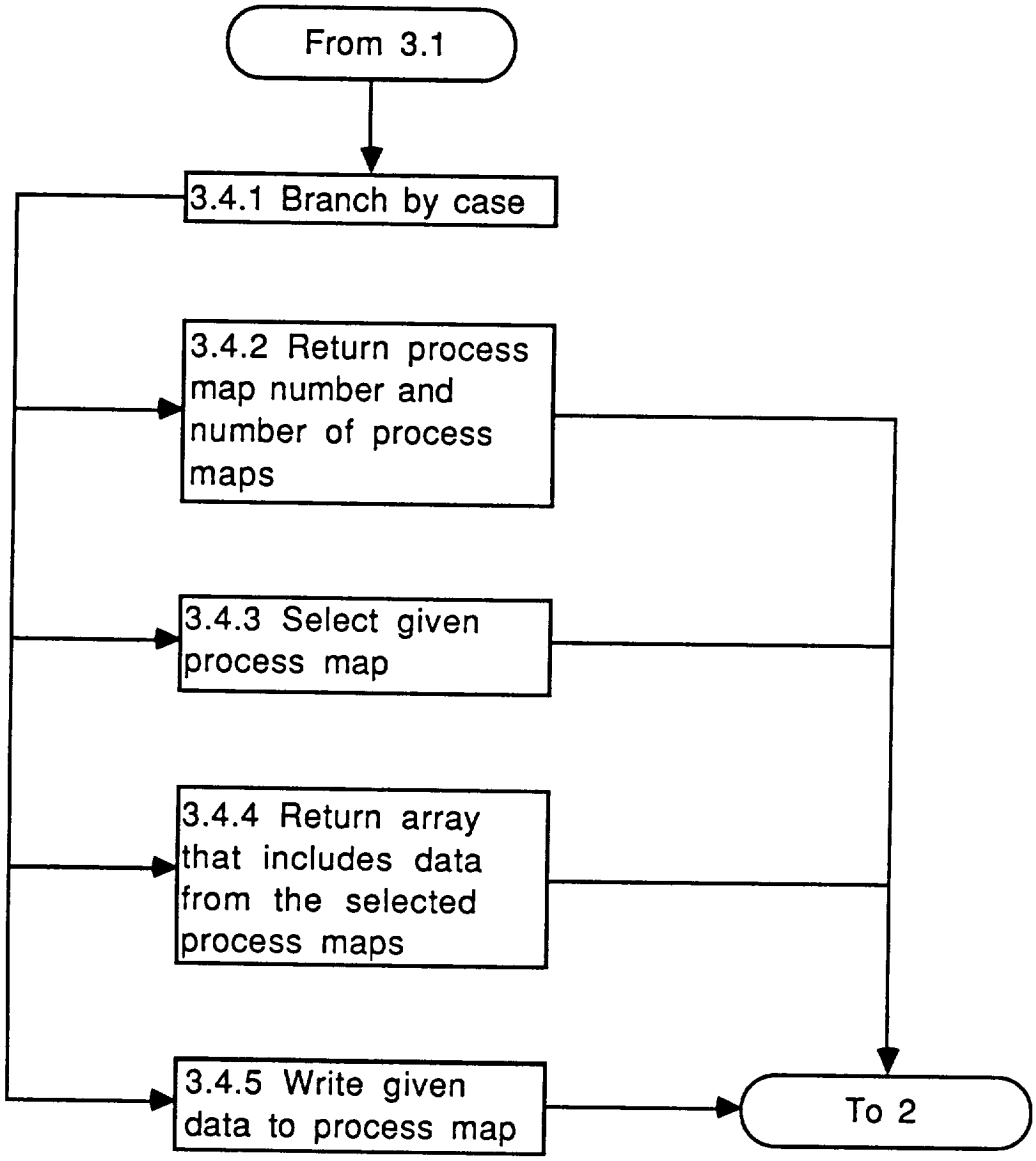
Sheet 3.2

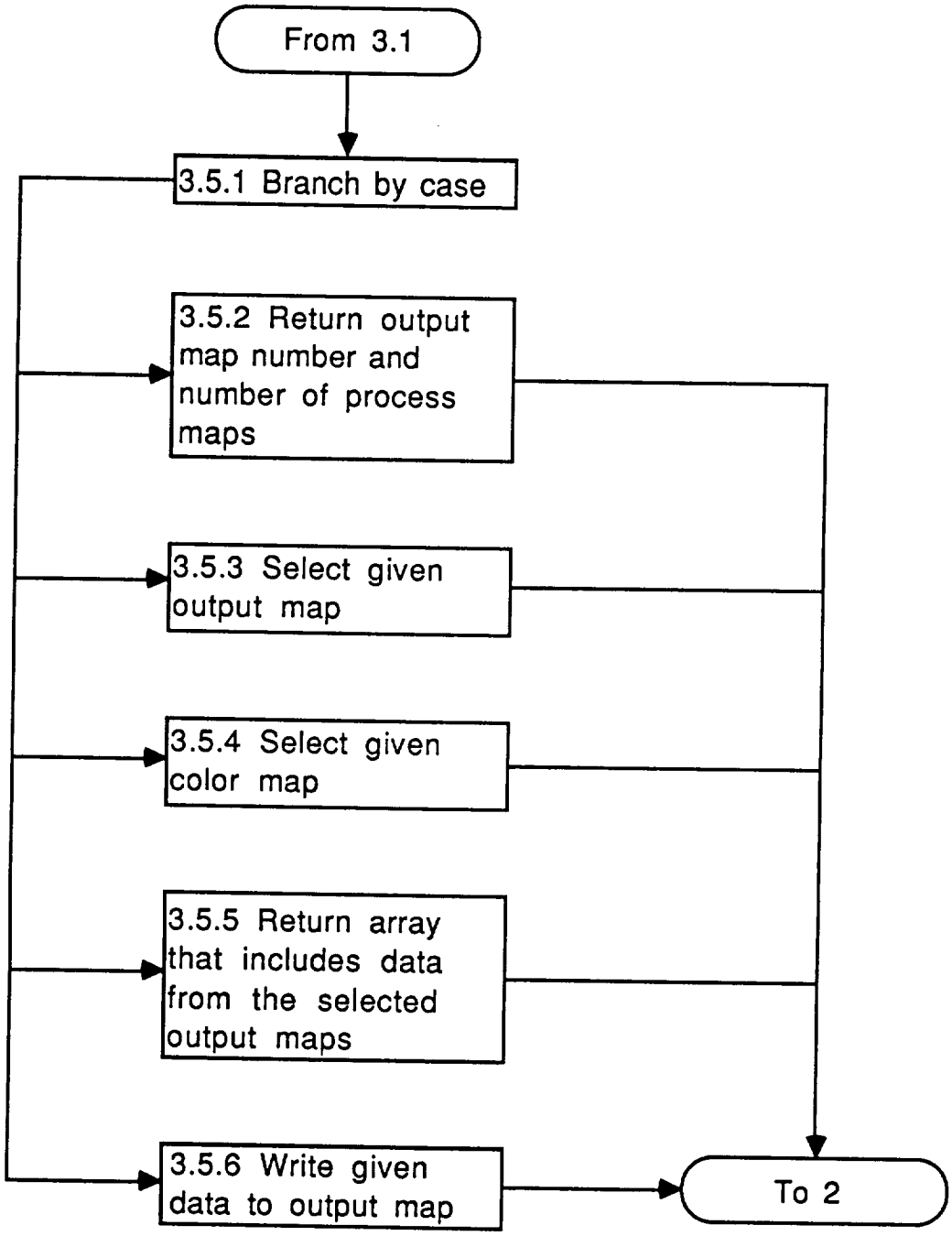

From 3.1
3.2.1 Branch by case
3.2.2 Setup default values for process maps, color maps, convolution banks, output maps, number of frames, camera, and frame size
3.2.3 Return value for number of frames
3.2.4 Return frame size
To 2
3.2.5 Exit PC software

Sheet 3.3

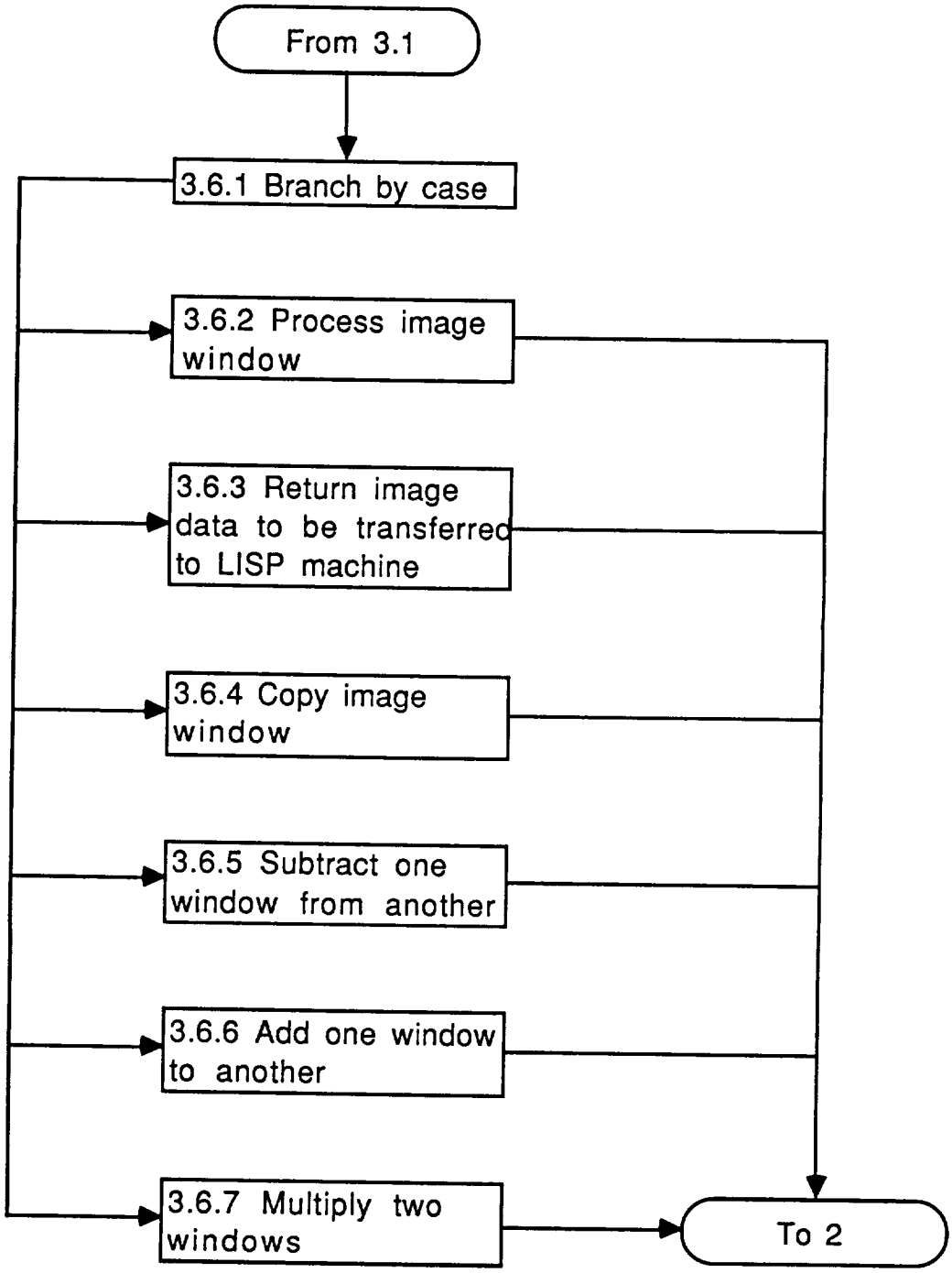

From 3.1
3.3.1 Branch by case
3.3.2 Select a given camera for input
3.3.3 Go to live mode
3.3.4 Snap a picture and store it in the selected frame buffer
3.3.5 Display the selected frame buffer
3.3.6 Turn display off
To 2

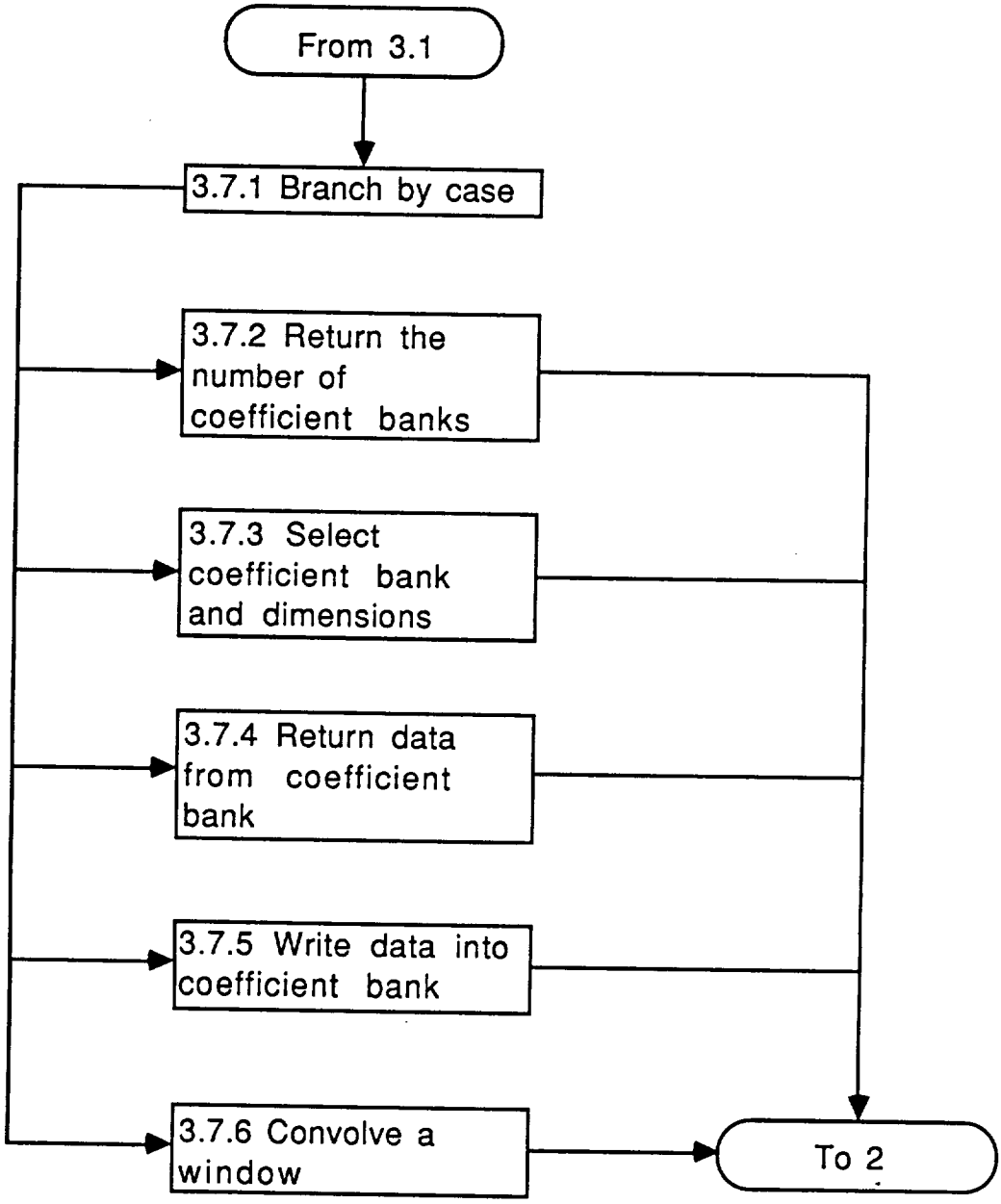
Sheet 3.4

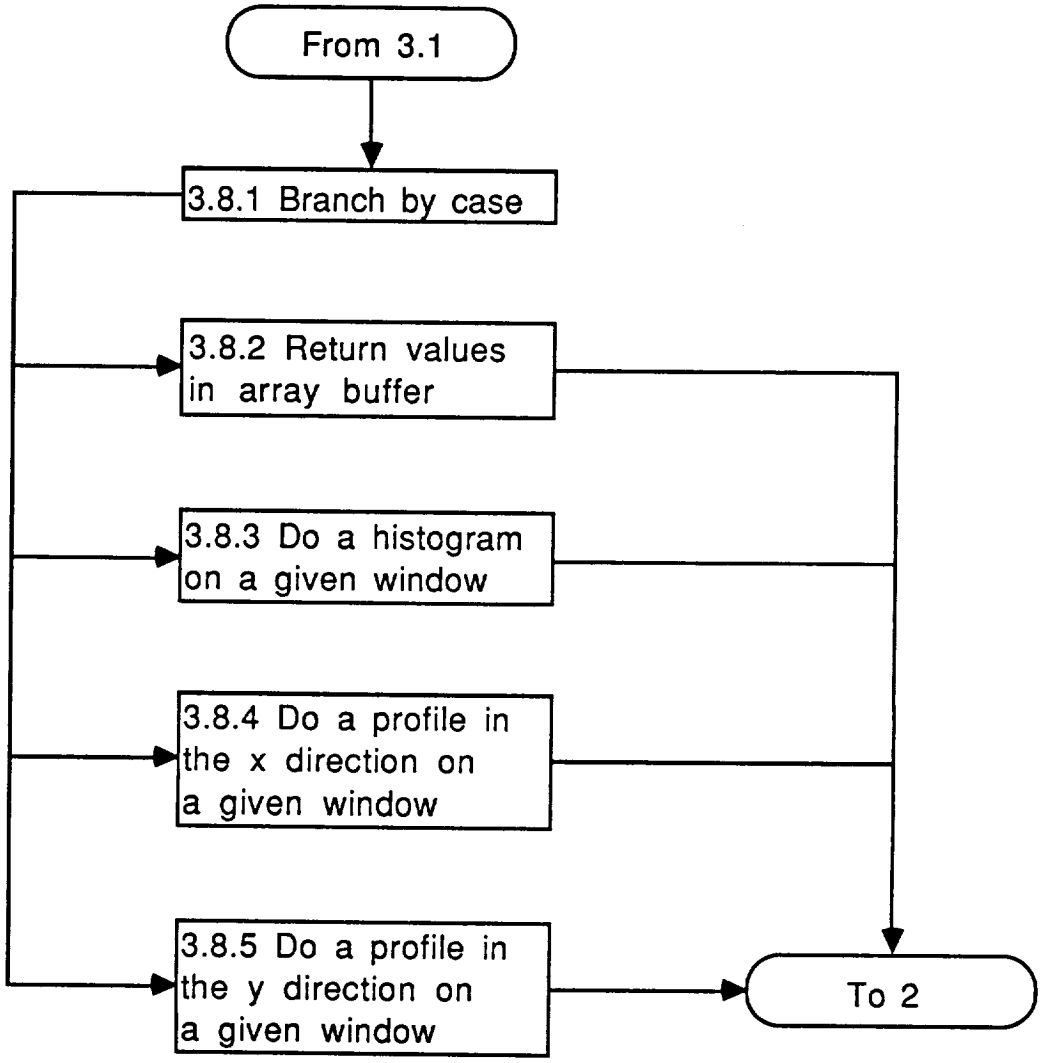

From 3.1
3.4.1 Branch by case
3.4.2 Return process map number and number of process maps
3.4.3 Select given process map
3.4.4 Return array that includes data from the selected process maps
3.4.5 Write given data to process map
To 2


From 3.1
3.5.1 Branch by case
3.5.2 Return output map number and number of process maps
3.5.3 Select given output map
3.5.4 Select given color map
3.5.5 Return array that includes data from the selected output maps
3.5.6 Write given data to output map
To 2

Sheet 3.6


From 3.1
3.6.1 Branch by case
3.6.2 Process image window
3.6.3 Return image data to be transferred to LISP machine
3.6.4 Copy image window
3.6.5 Subtract one window from another
3.6.6 Add one window to another
3.6.7 Multiply two windows
To 2


From 3.1
3.7.1 Branch by case
3.7.2 Return the number of coefficient banks
3.7.3 Select coefficient bank and dimensions
3.7.4 Return data from coefficient bank
3.7.5 Write data into coefficient bank
3.7.6 Convolve a window
To 2


From 3.1
3.8.1 Branch by case
3.8.2 Return values in array buffer
3.8.3 Do a histogram on a given window
3.8.4 Do a profile in the x direction on a given window
3.8.5 Do a profile in the y direction on a given window
To 2

Appendix D Source code for PC image processing software

D.1 Main program

```
/*****************************************************************/
/*                                                               */
/*            Symbolic Image Processing Software                 */
/*                                                               */
/*            7/7/1988 David Gilliam UAH                         */
/*                     and Shu Lam   UAH                         */
/*                                                               */
/*            with routines from rs232.c                         */
/*            11/16/1987 JZ                                      */
/*                                                               */
/*****************************************************************/

#include <debugat.h>
#include "rs232.h"
#include "def.h"
#include "exter.h"

struct image global;

main()
  {
  global.o[0] = '\0';
  getdat(&global);
  printf("\nSet\n");
  set_port(COM1, BD9600 + DATA8 + STOP1 + PARITY_N);
  while(TRUE)
    {
    global.command = getline(&global);
    printf("\n Command = %d\n",global.command);
    if (global.command==99) unknown();
    else
      {
      global.charcount = 0;
      resp0();
      switch(global.command)
       {
       case 1:  init(&global);      break;
       case 2:  framen(&global);    break;
       case 3:  frames(&global);    break;
       case 4:  selcam(&global);    break;
       case 5:  golive(&global);    break;
       case 6:  snapit(&global);    break;
       case 7:  dison(&global);     break;
       case 8:  disoff(&global);    break;
       case 9:  getnumpm(&global);  break;
       case 10: selpm(&global);     break;
       case 11: readpm(&global);    break;
       case 12: writpm(&global);    break;
```

```
            case 13: getnumom(&global);  break;
            case 14: selom(&global);     break;
            case 15: selcolmap(&global); break;
            case 16: readom(&global);    break;
            case 17: writomap(&global);  break;
            case 18: proc(&global);      break;
            case 19: trans(&global);     break;
            case 20: copy(&global);      break;
            case 21: sub(&global);       break;
            case 22: add(&global);       break;
            case 23: mult(&global);      break;
            case 24: getnumcb(&global);  break;
            case 25: selcb(&global);     break;
            case 26: readc(&global);     break;
            case 27: writc(&global);     break;
            case 28: conv(&global);      break;
            case 29: readv(&global);     break;
            case 30: hist(&global);      break;
            case 31: prox(&global);      break;
            case 32: proy(&global);      break;
            case 33: quit(&global);      break;
            case 34: test(&global);      break;
          }
          seend();
          }
      }
  }
```

D.2 Include file rs232.h

```
/* rs232.h */
/*
 * Definitions for communication port I/O.
 *
 *    11/16/1987 JZ
 */

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

#define     TIMEOUT     50000          /* Connection time-out */
#define     COM1        0x3F8          /* Port identification */
#define     COM2        0x2F8

#define     BD110       0x00           /* Available baud rates */
#define     BD150       0x20
#define     BD300       0x40
#define     BD600       0x60
#define     BD1200      0x80
#define     BD2400      0xA0
#define     BD4800      0xC0
#define     BD9600      0xE0

#define     PARITY_E    0x18           /* Port parity settings */
#define     PARITY_O    0x08
#define     PARITY_N    0x00

#define     STOP1       0x00           /* Number of stop-bits */
#define     STOP2       0x04

#define     DATA8       0x03       /* Data bits eight or seven */
#define     DATA7       0x02

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

extern int  set_port();              /* Set parameters of port */
extern int  tx();                    /* Send character over port */
extern int  txrdy();   /* Test if transmitter of port is ready */
extern int  rx();                   /* Receive character over port */
extern int  rxrdy();   /* Test if receiver of port is ready */
extern  int    musttx();    /*  Wait  for  tx  ready,  then  send
                          character*/
extern int  mustrx();          /* Wait for character from port */
```

D.3 Include file def.h

```
/* def.h */

#define NBANK 5
#define DIMENSN 49
#define FRAMEXSIZE 512
#define FRAMEYSIZE 480
#define NUMFRAMES 4
#define NUMOMAP 32
#define NUMPMAP 32
#define LINESIZE 2000
#define TRUE 1


struct image (char  names[50][10];
              int   parms[50];
              char  o[LINESIZE];
              char  word[50];
              int   i,j,command,numfunct,omnum,pmnum;
              int   parmlist[50],color;
              unsigned long buff[4098];
              char coef[NBANK][DIMENSN];
              int tcoef[NBANK][DIMENSN];
              char *ptr;
              int  abank,sidx;
              int xsize,ysize;
              unsigned char buf1[512],buf2[512],buf3[512];
              unsigned char buf4[300],buf5[300];
              int charcount;
                };
```

D.4 Include file exter.h

```
/* exter.h */


extern void init();
extern void framen();
extern void frames();
extern void selcam();
extern void golive();
extern void snapit();
extern void dison();
extern void disoff();
extern void getnumpm();
extern void selpm();
extern void readpm();
extern void writpm();
extern void getnumom();
extern void selom();
extern void selcolmap();
extern void readom();
extern void writomap();
extern void proc();
extern void trans();
extern void copy();
extern void sub();
extern void add();
extern void mult();
extern void getnumcb();
extern void selcb();
extern void readc();
extern void writc();
extern void conv();
extern void readv();
extern void hist();
extern void prox();
extern void proy();
extern void quit();
extern void test();
```

D.5 General image processing commands

```
/*****************************************************************/
/*                                                               */
/*   im.c                                                        */
/*                                                               */
/*   image processing functions                                  */
/*                                                               */
/*   This file includes everything on sheet 3.2                  */
/*                                                               */
/*****************************************************************/

#include "def.h"
#include "rs232.h"
#include <debugat.h>
#include "stdlib.h"

/*****************************************************************/
/*                                                               */
/*      G E N E R A L   I M A G E   F U N C T I O N S            */
/*                                                               */
/*  Initialize the image processor hardware system, clear up     */
/*    frame buffers.                                             */
/*  Returns #frames, x and y dimension of the frames in          */
/*    the system.                                                */
/*  Exit program                                                 */
/*                                                               */
/*     3.2.2  initialize                                         */
/*     3.2.3  get number of frames                               */
/*     3.2.4  get frame size                                     */
/*     3.2.5  quit                                               */
/*                                                               */
/*****************************************************************/

test()
  {
  int i=0,j=0;
  int nibel1,nibel2;
  char *p,c[20];

  printf("\nTest function\n");
  resp1();
  sendstr("20 20");
  for(i=0;i<20;i++)
    for(j=0;j<20;j++)
      {
      nibel1=j/0x10;
      nibel2=j%0x10;
      p=itoa(nibel1,c,0x10);
      musttx(COM1,c[0]);
      p=itoa(nibel2,c,0x10);
```

```
        musttx(COM1,c[0]);
        }
    printf("Done!!\n");
    musttx(COM1,' ');
    }

/* 3.2.2 initialize function */
init(p)
struct image *p;
  {
  printf("\nInitialize\n");
  im_init(0xd000,0x300);
  p->omnum = 0;
  p->pmnum = 0;
  p->color = 0;
  im_disformat(0,1,1);
  im_clear(4,0);
  im_clear(5,0);
  resp1();
  printf("\n");
 }


/* 3.2.3  get number of frames */
framen()
  {
  char temp[20];
  char *p;

  printf("\nGet Number Frames\n");
  resp1();
  p = itoa(NUMFRAMES,temp,10);     /*   convert integer to Ascii
                                   characters by using microsoft
                                   build-in function */
  printf("\nValue = %s\n",temp);
  sendstr(temp);
  }

/* 3.2.4 get frame size  */
frames()
  {
  char temp[20];
  char *p;

  printf("\nGet Frame Size\n");
  resp1();
  p = itoa(FRAMEXSIZE,temp,10);
  sendstr(temp);
  p = itoa(FRAMEYSIZE,temp,10);
  sendstr(temp);
  }
```

```
/*  3.2.5  exit program */
quit()
  {
  resp1();
  seend();
  printf("\nProgram exited\n");
  exit(0);
  }
```

D.6 Image acquisition functions

```
/*****************************************************************/
/*                                                               */
/*   acq.c                                                       */
/*                                                               */
/*   image processing functions                                  */
/*                                                               */
/*   This file includes everything on sheet 3.3                  */
/*                                                               */
/*****************************************************************/

#include "def.h"
#include "rs232.h"
#include <debugat.h>

/*****************************************************************/
/*                                                               */
/*              I M A G E    A C Q U I S T I O N                 */
/*          A N D   D I S P L A Y   F U N C T I O N S            */
/*                                                               */
/*     3.3.2 Select camera                                       */
/*     3.3.3 Live mode                                           */
/*     3.3.4 Snap a picture                                      */
/*     3.3.5 Display frame buffer                                */
/*     3.3.6 turn display off                                    */
/*                                                               */
/*****************************************************************/
/*---------------------------------------------------------------*/
/*   3.3.2  select camera                                        */
/*          Sets the selected or default camer input to port     */
/*          "number". This camera will be used for all the       */
/*          image acquisitions operations until another camera   */
/*          is selected.                                         */
 /*--------------------------------------------------------------*/


selcam(p)
struct image *p;
  {
  printf("\nCamera number %d selected\n",p->parmlist[0]);
  if (p->parmlist[0]<0 || p->parmlist[0]>3)
    {
    printf("\nError!! Camera number must be from 0 to 3\n");
    senderr();
    return(0);
    }
  im_chan(p->parmlist[0]);
  resp1();
  }
```

```
/*-------------------------------------------------------*/
/*  3.3.3 Puts image processor in the live mode.         */
/*-------------------------------------------------------*/

golive()
  {
  printf("\nSmile! You're on candid camera.\n");
  im_sync(1,0);
  im_video(0,0);
  resp1();
  }

/*-------------------------------------------------------*/
/*  3.3.4  Setup the channel and hardware configuration  */
/*         and take a snapshot then store the image to   */
/*         specified frame buffer.                       */
/*-------------------------------------------------------*/

snapit(p)
struct image *p;
  {
  printf("\nClick.\nPicture stored in buffer number
%d\n",p->parmlist[0]);
  if (p->parmlist[0]<0 || p->parmlist[0]>9)
    {
    printf("\nError!! Buffer number must be from 0 to 9");
    senderr();
    return(0);
    }
  im_opmode(2,p->parmlist[0]);
  im_outpath(p->parmlist[0],-1,0,0);
  im_inmode(1);
  im_sync(1,0);
  im_video(1,1);
  im_snapshot(p->parmlist[0]);
  resp1();
  }


/*-------------------------------------------------------*/
/*  3.3.5 display on                                     */
/*        Display the selected frame buffer by using     */
/*        MVP-AT command im_video () and im_outpath(),   */
/*        those two command enable display and select    */
/*        frame buffer to display.                       */
/*-------------------------------------------------------*/
dison(p)
struct image *p;
  {
  printf("\nImage number %d selected\n",p->parmlist[0]);
  if (p->parmlist[0]<0 || p->parmlist[0]>9)
    {
```

```
      printf("\nError!! Image number must be from 0 to 9");
      senderr();
      return(0);
      }
    im_video(1,1);
    im_opmode(2,p->parmlist[0]);
    im_outpath(p->parmlist[0],-1,0,0);
    resp1();
    }

/*-------------------------------------------------------------*/
/*  3.3.6  display off                                         */
/*         Turn off the display by using MVP-AT                */
/*         command im_video (0,1). 0:disable display           */
/*-------------------------------------------------------------*/
disoff()
  {
  printf("\nDisplay off\n");
  im_video(0,1);
  resp1();
  }
```

D.7 Process map functions

```
/*****************************************************************/
/*                                                               */
/*   proc.c                                                      */
/*                                                               */
/*   image processing functions                                  */
/*                                                               */
/*   This file includes everything on sheet 3.4                  */
/*                                                               */
/*****************************************************************/

#include "def.h"
#include "rs232.h"
#include <debugat.h>
#include "stdlib.h"

/*****************************************************************/
/*                                                               */
/*        P R O C E S S   M A P   F U N C T I O N S              */
/*                                                               */
/*    The following functions work with the input process map     */
/*    by  using MVP-AT command im_rlut(), im_wlut. Im_rlut()     */
/*    command reads a specified set of values from a lookup      */
/*    table color palette, and store them in a workbuffer.       */
/*    Im_wlut command writes the specified contents from a       */
/*    workbuffer, and stores them in a LUT color palette.        */
/*                                                               */
/*    3.4.2  get number of process map                           */
/*    3.4.3  select process map                                  */
/*    3.4.4  read process map                                    */
/*    3.4.5  write process map                                   */
/*                                                               */
/*****************************************************************/

/* 3.4.2 get number of process map  */
getnumpm(p)
struct image *p;
  {
  char *point;
  char temp[20];

  printf("\nGet number of process maps and map number\n");
  printf("\nProcess map %d selected out of %d
maps.",p->pmnum,NUMPMAP);
  resp1();
  point = itoa(p->pmnum,temp,10);
  sendstr(temp);
  point = itoa(NUMPMAP,temp,10);
  sendstr(temp);
  }
```

```
/*  3.4.3 select process map  */
selpm(p)
struct image *p;
  {
  printf("\nProces map number %d selected\n",p->parmlist[0]);
  if (p->parmlist[0]<1 || p->parmlist[0]>NUMPMAP)
    {
    printf("\nError!!  Process  map  number  must  be  from  1  to
%d",NUMPMAP);
    senderr();
    return(0);
    }
  im_slut(0,p->parmlist[0]-1); /* MVP-AT command which allows the
                                  user  to  select  a  specified
                                  lookup  table  palette  in  the
                                  input or output LUTs */
  p->pmnum = p->parmlist[0];
  resp1();
  }

/*  3.4.4 read process map  */
readpm(p)
struct image *p;
  {
  int i,num;
  char *point,temp[20];
  int  tidx,pmap;

   tidx = p->parmlist[0]; /* starting index  */
   num  = p->parmlist[1];   /*  number  of  cells  transfer  to  LISP
                                machine */
      if (num < 0 || num >256 || (tidx+num) > 256)
    {
      printf("\nERROR !! Incorrect cell number\n");
      senderr();
      return(0);
     }
   else
     {
       resp1();
       im_rlut(0,p->pmnum-1,tidx,num,p->buf4,p->buf4,p->buf4);
       for (i=tidx; i<num+tidx; i++)
       {
         pmap=p->buf4[i];
      printf("\n = %x\n",pmap);
      point=itoa(pmap,temp,0x10);
      sendstr(temp);
     }
      }
   }

/*  3.4.5 write process map  */
```

```
writpm(p)
struct image *p;
  {
  int i,num,tidx;

  tidx=p->parmlist[0]; /* starting index */
  num=p->parmlist[1];  /* number of cells to be transferred
                          from LISP machine */
  printf("\nWrite process map\n");
  if (num<1 || num>256)
    {
    printf("\nError!! Number of values written must be from 0 to
256\n");
    senderr();
    return(0);
    }
  if (tidx+num>256)
    {
    printf("\nError!! Values may not be written past end of
map\n");
    senderr();
    return(0);
    }
  for(i=tidx;i<tidx+num;i++) p->buf4[i] = p->parmlist[i+2];
  im_wlut(0,p->pmnum-1,tidx,num,p->buf4,p->buf4,p->buf4);
  resp1();
  }
```

D.8 Output map functions

```
/*****************************************************************/
/*                                                               */
/*   outp.c                                                      */
/*                                                               */
/*   image processing functions                                  */
/*                                                               */
/*   This file includes everything on sheet 3.5                  */
/*                                                               */
/*****************************************************************/

#include "def.h"
#include "rs232.h"
#include <debugat.h>
#include "stdlib.h"
#include "stdio.h"

/*****************************************************************/
/*                                                               */
/*          O U T P U T   M A P   F U N C T I O N S              */
/*                                                               */
/*    The following functions work with the output process map*/
/*    by  using MVP-AT command im_rlut(), im_wlut. Im_rlut()   */
/*    command reads a specified set of values from a lookup    */
/*    table color palette, and store them in a workbuffer.     */
/*    Im_wlut command writes the specified contents from a     */
/*    workbuffer, and stores them in a LUT color palette.      */
/*                                                               */
/*    3.5.2  get number of output maps                           */
/*    3.5.3  select output map                                   */
/*    3.5.4  select color map                                    */
/*    3.5.5  read output map                                     */
/*    3.5.6  write output map                                    */
/*                                                               */
/*****************************************************************/

/*  3.5.2  get number of output maps  */
getnumom(p)
struct image *p;
  {
  char *point;
  char temp[20];
  printf("\nGet number of output maps and map number\n");
  printf("\nOutput  map  %d  selected  out  of  %d
maps.",p->omnum,NUMOMAP);
  resp1();
  point = itoa(p->omnum,temp,10);
  sendstr(temp);
  point = itoa(NUMOMAP,temp,10);
  sendstr(temp);
  }
```

```
/*  3.5.3  select output map  */
selom(p)
struct image *p;
  {
  printf("\nOutput map number %d selected\n",p->parmlist[0]);
  if (p->parmlist[0]<1 || p->parmlist[0]>NUMOMAP)
    {
    printf("\nError!!  Output  map  number  must  be  from  1  to
%d",NUMOMAP);
    senderr();
    return(0);
    }
  im_slut(1,p->parmlist[0]-1);
  p->omnum = p->parmlist[0];
  resp1();
  }

/*  3.5.4  select color map  */
selcolmap(p)
struct image *p;
  {
  printf("\nColor map number %d selected\n",p->parmlist[0]);
  if (p->parmlist[1]<0 || p->parmlist[1]>2)
    {
    printf("\nError!! Color map number must be from 0 to 2\n");
    senderr();
    return(0);
    }
  p->color = p->parmlist[0];
  resp1();
  }

/*  3.5.5  read output map */
readom(p)
struct image *p;
  {
  int i,num,tidx,omap;
  char *point,temp[20];

  tidx = p->parmlist[0];  /*  starting indx   */
  num = p->parmlist[1];    /*  number of cells transfer to LISP
                               machine  */
  if(num <0 || num>256 || (tidx+num) > 256)
    {
      printf("\nERROR !! Incorrect cell number\n");
      senderr();
      return(0);
     }
  else
    {
      resp1();
```

```
        if (p->color = 0)      im_rlut
          (1,p->omnum-1,tidx,num,p->buf5,NULL,NULL);
        if (p->color = 1)      im_rlut
          (1,p->omnum-1,tidx,num,NULL,p->buf5,NULL);
        if (p->color = 0)      im_rlut
          (1,p->omnum-1,tidx,num,NULL,NULL,p->buf5);
        for (i=tidx; i<num+tidx; i++)
         {
           omap=p->buf5[i];
        printf("\n = %x\n",omap);
        point=itoa(omap,temp,0x10);
        sendstr(temp);
        }
        }
   }

/*  3.5.6  write output map  */
writomap(p)
struct image *p;
  {
  int i,tidx,num;

  tidx=p->parmlist[0];  /* starting index */
  num=p->parmlist[1];   /* number of cells to be transferred
                           from LISP machine */
  printf("\nWrite output map\n");
  if (num<1 || num>256)
    {
    printf("\nError!! Number of values written must be from 0 to
256\n");
    senderr();
    return(0);
    }
  if (tidx+num>256)
    {
    printf("\nError!! Values may not be written past end of
map\n");
    senderr();
    return(0);
    }
  for(i=tidx;i<tidx+num;i++) p->buf5[i]= p->parmlist[i+2];
  if (p->color = 0)
        im_wlut(1,p->omnum-1,tidx,num,p->buf5,NULL,NULL);
  if (p->color = 1)
        im_wlut(1,p->omnum-1,tidx,num,NULL,p->buf5,NULL);
  if (p->color = 2)
        im_wlut(1,p->omnum-1,tidx,num,NULL,NULL,p->buf5);
  resp1();
  }
```

D.9 Frame functions

```
/*****************************************************************/
/*                                                               */
/*   frame.c                                                     */
/*                                                               */
/*   image processing functions                                  */
/*                                                               */
/*   This file includes everything on sheet 3.6                  */
/*                                                               */
/*****************************************************************/


#include "def.h"
#include "rs232.h"
#include <debugat.h>
#include "stdlib.h"

/*****************************************************************/
/*                                                               */
/*            F R A M E   F U N C T I O N S                      */
/*                                                               */
/*    3.6.2 Process image                                        */
/*    3.6.3 Transfer image                                       */
/*    3.6.4 Copy image                                           */
/*    3.6.5 Add two image                                        */
/*    3.6.6 Subtract two image                                   */
/*    3.6.7 multiply two image                                   */
/*                                                               */
/*****************************************************************/

/*---------------------------------------------------------------*/
/*  3.6.2  Process image                                         */
/*         Pass the portion of the image through the selected    */
/*      process map and store the resultant image in             */
/*      destination window of frame buffer.                      */
/*         Using MVP-AT command im_pixblt()  enable us to        */
/*         copy data from the data included in the rectangle     */
/*         in the source frame buffer to a rectangle in the      */
/*         destination buffer.                                   */
/*---------------------------------------------------------------*/

/*---------------------------------------------------------------*/
/*    Each paramater passes to the following functions has its   */
/*  own meaning. ( except for function add, subtract, multiply)  */
/*           sbuf : source frame buffer                          */
/*           dbuf : destination frame buffer                     */
/*        x1,y1 : the upper left coordinate (x,y) of window-1    */
/*                  of source buffer                             */
/*        x2,y2 : the lower right coordinate (x,y) of window-1   */
/*                  of source buffer                             */
/*        x3,y3 : the upper left coordinate (x,y) of window-2    */
```

```
/*                    of destination buffer                          */
/*         x4,y4 : the lower right coordinate (x,y) of window-2      */
/*                    of destination buffer                          */
/*       dimx,dimy : dimension x, dimension y of the specified       */
/*                    window size                                    */
/*---------------------------------------------------------------*/

proc(p)
struct image *p;
  {
  int x1,y1,x2,y2,x3,y3,x4,y4,dimx,dimy,sbuf,dbuf;

  sbuf=p->parmlist[0];
  dbuf=p->parmlist[5];
  x1=p->parmlist[1]; y1=p->parmlist[2];
  x2=p->parmlist[3]; y2=p->parmlist[4];
  x3=p->parmlist[6]; y3=p->parmlist[7];
  x4=p->parmlist[8]; y4=p->parmlist[9];

  dimx = x4 - x3 + 1;
  dimy = y4 - y3 + 1;


/*--------     M O V E    A N    I M A G E      -----------*/

  printf("\nMove rectangle\n");
  im_opmode(2,0);
  im_pixblt(sbuf,x1,y1,x1+dimx,y1+dimy,dbuf,x3,y3,0,1);
  resp1();
  }


/*---------------------------------------------------------------*/
/*  3.6.3 Transfer image                                         */
/*        Return image data to be transferred to LISP machine*/
/*     The data will sent three blocks of ASCII characters:      */
/*     X-dimension, Y-dimension, and intensity information       */
/*     of each pixel of the portion of the image.                */
/*---------------------------------------------------------------*/

trans(p)
struct image *p;
  {
  int x1,y1,x2,y2,dimx,dimy,sbuf,i,j;
  int pixvlu;
  int nibel1,nibel2;
  char nib1,nib2;
  char *point,temp[20];

  sbuf=p->parmlist[0];
  x1=p->parmlist[1]; y1=p->parmlist[2];
  x2=p->parmlist[3]; y2=p->parmlist[4];
```

```
  dimx = x2 - x1 + 1;
  dimy = y2 - y1 + 1;

/*-------      T R A N S F E R    A N    I M A G E      ------*/

 if(x1>=0 && y1>=0 && x2>=0 && y2>=0 && dimx>=0 && dimy>=0
     && dimx<FRAMEXSIZE && dimy<FRAMEYSIZE)
     {
      printf("\nTransfer image\n");
      resp1();
      point = itoa(dimx,temp,10);
      sendstr(temp);
      point = itoa(dimy,temp,10);
      sendstr(temp);

      im_opmode(0,sbuf);
      for(i=0;i<dimy;i++)
      {
        im_rowr(x1,y1+i,dimx,p->buf1);  /*  MVP-AT command reads
                                        a user defined number of
                                        pixels one row at a time
                                        store  in  workbuffer
                                        buf1  */
        for(j=0;j<dimx;j++)
        {
           pixvlu=p->buf1[j];            /*  return value of each
                                             pixel  */
             nibel1=pixvlu/0x10;          /*  devided by hex 10
                                        receives  first  hex
                                        number */
        nibel2=pixvlu%0x10;           /*  module by hex 10 receives
                                       second hex number */

        /*  transmitt two nibels to LISP machine  */
        point=itoa(nibel1,temp,0x10);
        musttx(COM1,temp[0]);
        point=itoa(nibel2,temp,0x10);
        musttx(COM1,temp[0]);
      }
       }
      musttx(COM1,' ');
      printf("Done!\n");
    }
 else
   {
     senderr();
     return(0);
   }
}
```

```
/*-------------------------------------------------------------*/
/*  3.6.4 Copy image                                           */
/*        Copy a portion of an image of source buffer and      */
/*     store in destination window of frame buffer             */
/*        Using MVP-AT command im_pixblt()  enable us to       */
/*        copy data from the data included in the rectangle    */
/*        in the source frame buffer to a rectangle in the     */
/*        destination buffer.                                  */
/*-------------------------------------------------------------*/
copy(p)
struct image *p;
{

  int x1,y1,x2,y2,x3,y3,x4,y4,dimx,dimy;
  int winx1,winy1;
  int sbuf,dbuf;

  sbuf=p->parmlist[0]; dbuf=p->parmlist[5];
  x1=p->parmlist[1]; y1=p->parmlist[2];
  x2=p->parmlist[3]; y2=p->parmlist[4];
  x3=p->parmlist[6]; y3=p->parmlist[7];
  x4=p->parmlist[8]; y4=p->parmlist[9];

  dimx = x4 - x3 +1;
  dimy = y4 - y3 +1;
  winx1= x1 + dimx -1;
  winy1= y1 + dimy -1;


/*--------      C O P Y    A N    I M A G E        -----------*/


if(winx1>=0 && winx1<FRAMEXSIZE && winy1>=0 && winy1<FRAMEYSIZE
   && x3< FRAMEXSIZE && y3< FRAMEYSIZE )
   {
      printf("\nCopy\n");
      im_opmode(2,sbuf);
      im_pixblt(sbuf,x1,y1,x1+dimx,y1+dimy,dbuf,x3,y3,0,1);
      resp1();
    }
else
  {
    senderr();
    printf("\nInvalid data, please check your data and try again
!!\n");
    return(0);
   }
  resp1();
 }

/*-------------------------------------------------------------*/
/*  This function uses the MVP-AT build-in interimage command  */
```

```
/*  to perform subtraction. This command works only when the   */
/*  upper left coordinate (x,y) of the window of source buffer */
/*  and destination buffer are the same, otherwise using our   */
/*  own implemention.                                          */
/*-------------------------------------------------------------*/

subtraction(x1,y1,dimx,dimy,sbuf1,sbuf2,dbuf)
  int x1,y1,dimx,dimy,sbuf1,sbuf2,dbuf;
  {
   printf("\nSubtract image\n");
   im_opmode(2,0);
   im_procwin(x1,y1,x1+dimx,y1+dimy);
   im_interimage(sbuf1,sbuf2,dbuf,1);
   im_procwin(0,0,512,480);
   }

/*-------------------------------------------------------------*/
/*  3.6.5 Subtract two images                                  */
/*        Subtract one window from another.                    */
/*        Subtract portions of an image in two source buffer   */
/*        and place the difference in the destination window   */
/*        of frame buffer.                                     */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*     For add, subtract, multiply functions :                 */
/*          sbuf1 : source frame buffer 1                      */
/*          sbuf2 : source frame buffer 2                      */
/*      x1,y1 : the upper left coordinate (x,y) of window-1    */
/*                  of source buffer 1                         */
/*      x2,y2 : the upper left coordinate (x,y) of window-2    */
/*                  of source buffer 2                         */
/*      x3,y3 : the upper left coordinate (x,y) of window-3    */
/*                  of destination buffer                      */
/*      x4,y4 : the lower right coordinate (x,y) of window-3   */
/*                  of destination buffer                      */
/*      dimx,dimy : dimension x, dimension y of the specified  */
/*                  window size                                */
/*      winx,winy : specifies window size of sbuf1 & sbuf2     */
/*                  to do image processing                     */
/*-------------------------------------------------------------*/
sub(p)
struct image *p;
 {
  int i,j,dimx,dimy;
  int x1,y1,x2,y2,x3,y3,x4,y4;
  int winx1,winy1,winx2,winy2;
  int sbuf1,sbuf2,dbuf;

  sbuf1=p->parmlist[0];
  sbuf2=p->parmlist[5];
  dbuf =p->parmlist[10];
```

```
   x1=p->parmlist[1];   y1=p->parmlist[2];
   x2=p->parmlist[6];   y2=p->parmlist[7];
   x3=p->parmlist[11];  y3=p->parmlist[12];
   x4=p->parmlist[13];  y4=p->parmlist[14];

   dimy = y4-y3+1;
   dimx = x4-x3+1;

   winx1= x1+dimx-1; winy1= y2+dimy-1;
   winx2= x2+dimx-1; winy2= y2+dimy-1;


/*------    S U B T R A C T    T W O    I M A G E    -------*/

if(x1>=0 && winx1<FRAMEXSIZE && y1>=0 && winy1<FRAMEYSIZE
    && x2>=0 && winx2<FRAMEXSIZE && y2>=0 &&winy2<FRAMEYSIZE
    && x3>=0 && y3>=0 && x4< FRAMEXSIZE && y4< FRAMEYSIZE )
     if((x1==x2 && x2==x3) && (y1==y2 && y2==y3))
           subtraction(x1,y1,dimx,dimy,sbuf1,sbuf2,dbuf);
     else
        {
    /* our own implementation of subtract function, using MVP-AT
       command im_colr() to read each pixel in a column and push
       them to workbuffer, im_colw() to write each column from
       workbuffer to the specified coordinate (x,y) of frame
buffer  */

       printf("\nSubtract two images\n");
       for(i=0;i<dimx;i++)
       {
        im_opmode(0,sbuf1);
        im_colr(x1+i,y1,dimy,p->buf1);
        im_opmode(0,sbuf2);
        im_colr(x2+i,y2,dimy,p->buf2);
        for(j=0;j<dimy;j++)
        p->buf3[j] = p->buf1[j] - p->buf2[j];
        im_opmode(0,dbuf);
        im_colw(x3+i,y3,dimy,p->buf3);
        }
  }
   else
  {
   senderr();
   printf("\nInvalid data, please check your data and try again
!!\n");
   return(0);
   }
 resp1();
 }

/*-----------------------------------------------------------*/
```

```
/*  This function uses the MVP-AT build-in interimage command */
/*  to perform addition. This command works only when the     */
/*  upper left coordinate (x,y) of the window of source buffer*/
/*  and destination buffer are the same, otherwise using our  */
/*  own implementation.                                       */
/*-----------------------------------------------------------*/
addition(x1,y1,dimx,dimy,sbuf1,sbuf2,dbuf)
  int x1,y1,dimx,dimy,sbuf1,sbuf2,dbuf;
  {
   printf("\nAdd image\n");
   im_opmode(2,0);
   im_procwin(x1,y1,x1+dimx,y1+dimy);
   im_interimage(sbuf1,sbuf2,dbuf,0);
   im_procwin(0,0,512,480);
 }


/*-----------------------------------------------------------*/
/*  3.6.6 Add two image                                      */
/*        Add one window to anothe.r                         */
/*         Add portions of images in two source frame buffer */
/*    and place the sum in destination window of frame buffer. */
/*-----------------------------------------------------------*/
 add(p)
 struct image *p;
  {
  int i,j,dimx,dimy;
  int x1,y1,x2,y2,x3,y3,x4,y4;
  int winx1,winy1,winx2,winy2;
  int sbuf1,sbuf2,dbuf;

  sbuf1=p->parmlist[0];
  sbuf2=p->parmlist[5];
  dbuf =p->parmlist[10];

  x1=p->parmlist[1];   y1=p->parmlist[2];
  x2=p->parmlist[6];   y2=p->parmlist[7];
  x3=p->parmlist[11];  y3=p->parmlist[12];
  x4=p->parmlist[13];  y4=p->parmlist[14];

  dimy = y4-y3+1;
  dimx = x4-x3+1;

  winx1= x1+dimx-1; winy1= y2+dimy-1;
  winx2= x2+dimx-1; winy2= y2+dimy-1;


/*------      A D D   T W O   I M A G E      ----------*/

if(x1>=0 && winx1<FRAMEXSIZE && y1>=0 && winy1<FRAMEYSIZE
    && x2>=0 && winx2<FRAMEXSIZE && y2>=0 &&winy2<FRAMEYSIZE
    && x3>=0 && y3>=0 && x4< FRAMEXSIZE && y4< FRAMEYSIZE )
```

```
      if((x1==x2 && x2==x3) && (y1==y2 && y2==y3))
            addition(x1,y1,dimx,dimy,sbuf1,sbuf2,dbuf);
      else
   {
     /* our own implementation of addition function, using MVP-AT
        command im_colr() to read each pixel in a column and push
        them to workbuffer, im_colw() to write each column from
        workbuffer to the specified coordinate (x,y) of frame
buffer  */

        printf("\nAdd two images\n");
        for(i=0;i<dimx;i++)
        {
         im_opmode(0,sbuf1);
         im_colr(x1+i,y1,dimy,p->buf1);
         im_opmode(0,sbuf2);
         im_colr(x2+i,y2,dimy,p->buf2);
         for(j=0;j<dimy;j++)
         p->buf3[j] = p->buf1[j] + p->buf2[j];
         im_opmode(0,dbuf);
         im_colw(x3+i,y3,dimy,p->buf3);
         }
   }
else
   {
    senderr();
    printf("\nInvalid data, please check your data and try again
!!\n");
    return(0);
    }
 resp1();
 }


/*-------------------------------------------------------------*/
/*  3.6.7 multiply two image                                   */
/*   multiply portions of inages in two source frame buffer    */
/* and place the product in destination window of frame buffer*/
/*-------------------------------------------------------------*/

mult(p)
struct image *p;
 {
  int i,j,dimx,dimy;
  int x1,y1,x2,y2,x3,y3,x4,y4;
  int winx1,winy1,winx2,winy2;
  int sbuf1,sbuf2,dbuf;

  sbuf1=p->parmlist[0];
  sbuf2=p->parmlist[5];
  dbuf =p->parmlist[10];
```

```
  x1=p->parmlist[1];   y1=p->parmlist[2];
  x2=p->parmlist[6];   y2=p->parmlist[7];
  x3=p->parmlist[11];  y3=p->parmlist[12];
  x4=p->parmlist[13];  y4=p->parmlist[14];

  dimy = y4-y3+1;
  dimx = x4-x3+1;

  winx1= x1+dimx-1; winy1= y2+dimy-1;
  winx2= x2+dimx-1; winy2= y2+dimy-1;


/*-------   M U L T I P L Y   T W O   I M A G E   -------*/

if(x1>=0 && winx1<FRAMEXSIZE && y1>=0 && winy1<FRAMEYSIZE
    && x2>=0 && winx2<FRAMEXSIZE && y2>=0 &&winy2<FRAMEYSIZE
    && x3>=0 && y3>=0 && x4< FRAMEXSIZE && y4< FRAMEYSIZE )
    {
    /* our own implementation of multiply function, using MVP-AT
       command im_colr() to read each pixel in a column and push
       them to workbuffer, im_colw() to write each column from
       workbuffer to the specified coordinate (x,y) of frame
  buffer  */

       printf("\nMultiply two images\n");
       for(i=0;i<dimx;i++)
       {
        im_opmode(0,sbuf1);
        im_colr(x1+i,y1,dimy,p->buf1);
        im_opmode(0,sbuf2);
        im_colr(x2+i,y2,dimy,p->buf2);
        for(j=0;j<dimy;j++)
        p->buf3[j] = p->buf1[j] * p->buf2[j];
        im_opmode(0,dbuf);
        im_colw(x3+i,y3,dimy,p->buf3);
        }
     }
 else
  { senderr();
   printf("\nInvalid data, please check your data and try again
!!\n");
   return(0);
   }
 resp1();
 }
```

D.10 Convolution functions

```
/*****************************************************************/
/*                                                               */
/*   conv.c                                                      */
/*                                                               */
/*   image processing functions                                  */
/*                                                               */
/*   This file includes everything on sheet 3.7                  */
/*                                                               */
/*****************************************************************/

#include "def.h"
#include "rs232.h"
#include <debugat.h>
#include "stdlib.h"

/*****************************************************************/
/*                                                               */
/*          C O N V O L U T I O N   F U N C T I O N S            */
/*                                                               */
/*   3.7.2 Get number of coefficient banks                       */
/*   3.7.3 Select coefficient bank                               */
/*   3.7.4 Read coefficients                                     */
/*   3.7.5 Write coefficients                                    */
/*   3.7.6 convolve image                                        */
/*                                                               */
/*****************************************************************/

/*---------------------------------------------------------------*/
/*   3.7.2 Get number of coefficient banks                       */
/*         Return the number of coefficient banks                */
/*         and their dimension                                   */
/*---------------------------------------------------------------*/

getnumcb()
  {
  char temp[20];
  char *point;

  printf("\nGet number of bank\n");
  printf("\nnumber-of-banks is %d",NBANK);
  printf(",  dimension is %d\n",DIMENSN);
  resp1();
  point=itoa(NBANK,temp,10);
  sendstr(temp);
  point=itoa(DIMENSN,temp,10);
  sendstr(temp);
  }

/*---------------------------------------------------------------*/
```

```
/*   3.7.3  Select coefficient bank and kernel size            */
/*      Select one of the banks from NBANK and descide kernel */
/*       size, kernel cannot over 7 x 7 dimension.             */
/*       All convolutions will use the kernel size and         */
/*       coefficient information from the selected bank.       */
/*-------------------------------------------------------------*/

selcb(p)
struct image *p;
{
  int selbank;
  int klsize,x,y;
  char temp[20];
  char *point;

  selbank=p->parmlist[0];
  x=p->parmlist[1];
  y=p->parmlist[2];

  if( selbank<0 || selbank>=NBANK)
    { printf("Selected bank is not available, current bank is
#0\n");
      p->abank = 0;
      }
  else
     p->abank = selbank ;

  klsize = x * y;
  if (klsize <0 || klsize> DIMENSN)
     { senderr();
       printf("\nkernel size can not over dimension 7 x 7 \n");
       return(0);
       }
  else
    { p->xsize=x;
      p->ysize=y;
      printf("\nSelect bank  # %d", p->abank );
      printf(",  kernel size = %d", p->xsize);
      printf(" x %d\n", p->ysize);
      resp1();
      point=itoa(NBANK,temp,10);
      sendstr(temp);
      point=itoa(p->xsize,temp,10);
      sendstr(temp);
      point=itoa(p->ysize,temp,10);
      sendstr(temp);
   }

}

/*-------------------------------------------------------------*/
/*   3.7.4  Read coefficients                                  */
```

```
/*          Read data from the coefficient array of the      */
/*          selected bank, read whatever in the coefficient  */
/*          array. It's better write data to the array first */
/*          then read it, otherwise you may read unexpected  */
/*          data.                                            */
/*-----------------------------------------------------------*/

readc(p)
struct image *p;
{
int i,num,acell;
char temp[20];
char *point;
int tidx;

tidx=p->parmlist[0]; /* starting index  */
num=p->parmlist[1]; /* number of cells transfer to LISP machine*/

printf("\nRead coefficent\n");
printf("\nActive bank data are : ");
if(num <0)
 { printf("Error !! Nagative cell number \n");
   senderr();
   return(0);}
else if (num > DIMENSN)
   { printf("Warning !! too many cells, display to the end of
data \n");
     resp1();
     for( i=tidx; i<DIMENSN; i++)
      {acell=p->coef[p->abank][i];
       printf(" %d",acell);
       point=itoa(acell,temp,10);
         sendstr(temp); }
       printf("\n");
    }
    else
      {
       resp1();
       for( i=tidx; i<num+tidx; i++)
       {acell=p->coef[p->abank][i];
       printf(" %d",acell);
       point=itoa(acell,temp,10);
       sendstr(temp);}
       printf("\n");
    }
}

/*-----------------------------------------------------------*/
/*   3.7.5  Write coefficients                               */
/*      Transfers data from Lisp machine then  write to      */
/*       coefficient array of the selected bank.              */
/*      Specifies  #cells need to be transferred, starts from*/
```

```
/*      the starting-index.                                          */
/*-------------------------------------------------------------------*/

writc(p)
struct image *p;
  {
   int i,j,num,cnt;

    cnt=p->parmlist[1]+1;
    for (i=2;i<=cnt;i++)
     { if (p->parmlist[i] > 127 || p->parmlist[i] < -127)
      { printf("\nInvalid data, kernel value out of range !!
\n");
        senderr();
        return(0);
      }
       else
        i++;
       }

    printf("\nWrite coefficient\n");

    num=p->parmlist[1];
    i=2;
    p->sidx=p->parmlist[0];
    for(j=p->sidx;j<num+p->sidx;j++)
    {
      p->coef[p->abank][j]=p->parmlist[i];
      p->tcoef[p->abank][j]=p->parmlist[i];
      i++;
    }
   resp1();

  }

/*-------------------------------------------------------------------*/
/*   This function uses the MVP-AT build-in convolve command         */
/*   to do the convolution. If the upper left coordinate (x,y)       */
/*   of the window of source buffer and destination buffer           */
/*   are the same, using build-in command instead of writing         */
/*   your own. A_convolution function is much faster than            */
/*   B_convolution function.                                         */
/*-------------------------------------------------------------------*/

  a_convolution (x1,y1,winx,winy,sbuf,dbuf,p)
  struct image *p;
  int x1,y1,winx,winy;
  int sbuf,dbuf;
 {     printf("\nConvolution\n");
       im_opmode(2,sbuf);
       im_procwin(x1,y1,x1+winx,y1+winy);
       p->ptr = &p->coef[p->abank][p->sidx];
```

```
        im_convolve(p->xsize,p->ysize,p->ptr,sbuf,dbuf);
        im_procwin(0,0,512,480);
 }

/*-------------------------------------------------------------*/
/*   B_convolution function is performmed only when the        */
/*   starting point (upper left coordinate x,y) of the         */
/*   window of source buffer and destination buffer are        */
/*   different. This function implement the convolution        */
/*   by reading each neighboring pixel of the current pixel    */
/*   through math calculating, then store the new pixel        */
/*   value back to the specified destination coordinate        */
/*   of the frame buffer.                                      */
/*-------------------------------------------------------------*/

  b_convolution(x1,y1,x2,y2,x3,y3,sbuf,dbuf,p)
    struct image *p;
    int x1,y1,x2,y2,x3,y3;
    int sbuf,dbuf;
    {
    int ux,uy;
    int dimx,dimy;
    int rem1,rem2,n,m,i,j,k,l,pn;
    int klsize;
    unsigned long temp,sum;
    int dcoef[DIMENSN];


    dimx=x3-x2+1;
    dimy=y3-y2+1;

    printf("\nConvolve test\n");
    klsize=p->xsize*p->ysize;
    rem1=p->xsize%2;
    rem2=p->ysize%2;
    if(rem1=0)
      n=p->xsize/2-1;
    else
      n=p->xsize/2;

    if (rem2=0)
       m=p->ysize/2-1;
    else
       m=p->ysize/2;

    j=p->sidx;
    for(i=0;i<klsize;i++)
       dcoef[i]=p->tcoef[p->abank][j++];

    for(j=0;j<dimx;j++)
       for(i=0;i<dimy;i++)
       {
```

```
        ux=x1+j; uy=y1+i;
      ux=ux-n; uy=uy-m; /* try to find starting coordinate of
                           n x m kernel */
      sum=0;
      pn=0;
      im_opmode(0,sbuf);
      for (k=0;k<p->ysize;k++)
        for (l=0;l<p->xsize;l++)
          {
           temp = im_pixr(ux+l,uy+k);
           sum = sum + (temp*dcoef[pn++]);
           }
      im_opmode(0,dbuf);
      im_pixw(x2+j,y2+i,sum);
      }
}

/*-------------------------------------------------------------*/
/*  3.7.6  Convolve function                                   */
/*         Convolve a portion of an image inside window-1 of*/
/*         frame buffer 1 and place results in window-2 of  */
/*         frame buffer 2 using the coefficients in the     */
/*         coefficient array.                                 */
/*-------------------------------------------------------------*/

 conv(p)
 struct image *p;
  {
  int i,winx,winy,x1,y1,x2,y2,x3,y3;
  int sbuf,dbuf;
  int n1,n2,n3;

  x1=p->parmlist[1]; y1=p->parmlist[2];
  x2=p->parmlist[6]; y2=p->parmlist[7];
  x3=p->parmlist[8]; y3=p->parmlist[9];
  winx=x3-x2;
  winy=y3-y2;
  sbuf=p->parmlist[0]; dbuf=p->parmlist[5];


  /*--------        C O N V O L U T I O N        -----------*/

  if(((sbuf==0 || sbuf==1) && dbuf==5) || ((sbuf==2 || sbuf==3)
&&
    dbuf==4))
      if(x1>=0 && y1>=0 && x3<=FRAMEXSIZE && y3<=FRAMEYSIZE)
         if((x1==x2) && (y1==y2))
              a_convolution(x1,y1,winx,winy,sbuf,dbuf,p);
      else
           b_convolution(x1,y1,x2,y2,x3,y3,sbuf,dbuf,p);
       else
          {
```

```
                senderr();
                return(0);
               }
 else
      {
       senderr();
       return(0);
      }

resp1();
}
```

D.11 Picture measurement functions

```
/*****************************************************************/
/*                                                               */
/*   meas.c                                                      */
/*                                                               */
/*   image processing functions                                  */
/*                                                               */
/*   This file includes everything on sheet 3.8                  */
/*                                                               */
/*****************************************************************/

#include "def.h"
#include "rs232.h"
#include <debugat.h>
#include "stdlib.h"

 /*****************************************************************/
 /*                                                               */
 /*          P I C T U R E    M E A S U R E M E N T               */
 /*                   F U N C T I O N S                           */
 /*                                                               */
 /*    Using MVP-AT build-in function im_histo(),                 */
 /*    im_profile() to do the following functions:                */
 /*    3.8.2  read values                                         */
 /*    3.8.3  histogram                                           */
 /*    3.8.4  profile-x                                           */
 /*    3.8.5  profile-y                                           */
 /*                                                               */
 /*****************************************************************/

/*  3.8.2  read values   */
readv(p)
struct image *p;
  {
   int i,num;
   char temp[20],*point;
   int tidx,hist;

   tidx = p->parmlist[0];
   num = p->parmlist[1];

   if (num < 0)
    {
      printf("\nERROR !! Negative cell number\n");
      senderr();
      return(0);
     }
   else
     {
       resp1();
       for (i=tidx; i<num+tidx; i++)
```

```
        {
          resp1();
          hist=p->buff[i];
        printf(" hist = %x",hist);
        point=itoa(hist,temp,0x10);
        sendstr(temp);
         }
        }
    }

/*   3.8.3  histogram     */
hist(p)
struct image *p;
  {int i;
  printf("\nGet histogram\n");
  im_opmode(2,p->parmlist[0]);
  im_procwin(p->parmlist[1],p->parmlist[2],p->parmlist[3],
     p->parmlist[4]);
  im_histo(p->parmlist[0],p->buff);
  im_procwin(0,0,512,480);
  resp1();
  }

/*  3.8.4  profile x     */
prox(p)
struct image *p;
  {
  printf("\nGet x-profile\n");
  im_opmode(2,p->parmlist[0]);
  im_procwin(p->parmlist[1],p->parmlist[2],p->parmlist[3],
     p->parmlist[4]);
  im_profile(0,1,p->parmlist[0],p->buff);
  im_procwin(0,0,512,480);
  resp1();
  }

/*  3.8.5  profile y      */
proy(p)
struct image *p;
  {
  printf("\nGet y-profile\n");
  im_opmode(2,p->parmlist[0]);
  im_procwin(p->parmlist[1],p->parmlist[2],p->parmlist[3],
     p->parmlist[4]);
  im_profile(2,1,p->parmlist[0],p->buff);
  im_procwin(0,0,512,480);
  resp1();
  }
```

D.12 Communication functions

```
/* rs2.c communications code */

#include "stdlib.h"
#include "stdio.h"
#include "rs232.h"
#include "def.h"
#include <dos.h>


/* load all function names from disk */
getdat(p)
struct image *p;
  {
  FILE *dat;
  char temp[3];
  int i=0,j=0;

  if ((dat = fopen("funct.dat","r")) == NULL)
    {
    perror("Error opening funct.dat");
    exit(-1);
    }
  while(fscanf(dat,"%s\n%s\n",p->names[i],temp)!=EOF)
   {
   p->parms[i] = atoi(temp);
   i++;
   }
  fclose(dat);
  p->numfunct = i-1;
  }

/* 2.5 Send error code */
unknown()
  {
  printf("\nUnkown command\n");
  senderr();
  return(0);
  }

resp1()
  {
  wait();
  musttx(COM1,'1');
  musttx(COM1,' ');
  }

senderr()
  {
  wait();
```

```
  musttx(COM1,'2');
  musttx(COM1,' ');
  }

seend()
  {
  musttx(COM1,'E');
  musttx(COM1,'N');
  musttx(COM1,'D');
  musttx(COM1,' ');
  }

resp0()
  {
  wait();
  musttx(COM1,'0');
  musttx(COM1,' ');
  seend();
  }

wait()
  {
  while(!txrdy(COM1));
  }

getline(p)
struct image *p;
  {
  int i,j,k=0;
  while(!rxrdy(COM1));
  i = 0;
  while((p->o[i-5]!=' ' || p->o[i-4]!='E' || p->o[i-3]!='N' ||
p->o[i-2]!='D' || p->o[i-1]!=' ')&&(i<LINESIZE))
    {
    p->o[i] = mustrx(COM1);
    i++;
    }
  printf("\nInfo recieved\n");
  if (i==LINESIZE)
    {
    printf("\nString recieved is greater than %d
characters\n",LINESIZE);
    printf("\n%d\n",i);
    return(99);
    }
  getword(p);
  j = 0;
  while(strcmp(p->word,p->names[j]) && (j <= p->numfunct))
    {
    printf("\n%d %s %s",j,p->word,p->names[j]);
    j++;
    }
```

```
    j++;
    if(j > p->numfunct) j = 99;
    if(j != 99)
      {
      getword(p);
      k=0;
      while(strcmp(p->word,"END"))
        {
        p->parmlist[k] = atoi(p->word);
        k++;
        getword(p);
        }
      printf("parms %d   expected %d",k,p->parms[j-1]);
      if(p->parms[j-1] != -1)
        {
        if(k != p->parms[j-1]) j=99;
        }
      else
        {
        if(p->parmlist[1] != k - 2)
          {
       printf("\nparms %d  expected list %d\n",k,p->parmlist[1]);
       j=99;
       }
        }
      }
    return(j);
    }

/* Get word from o[] strings and put it in word[] */
getword(p)
struct image *p;
    {
    int i,j=0;
    while(p->o[0]==' ')
      for(j=0;j<LINESIZE-1;j++)
        p->o[j]=p->o[j+1];
    j = 0;
    while(p->o[j]!=' ')
      {
      p->word[j]=p->o[j];
      j++;
      }
    p->word[j] = '\0';
    for(i=0;i<LINESIZE-j-1;i++)
      p->o[i]=p->o[i+j];
    printf("\nWord = %s\n",p->word);
    }

/* Send string of characters to COM1 */
sendstr(str)
   char str[50];
```

```
    {
    int i=0;
    while(str[i]!='\0' && i<50)
      {
      musttx(COM1,str[i]);
      i++;
      }
    musttx(COM1,' ');
    }

/* Set baud rate,#data bits,#stopbits,parity type */
set_port(port,mode)
    int port,mode;{
    union REGS r;
    r.h.ah=0; r.h.al=mode; r.x.dx=0; if(port==COM2) r.x.dx=1;
    int86(0x14,&r,&r);
    }

/* See if transmitter ready */
int txrdy(port)
    int port;{
    return((int)inp(port+5)&0x60);
    }

/* See if receiver ready */
int rxrdy(port)
    int port;{
    return((int)inp(port+5)&0x01);
    }

/* Send a character; transmitter must be empty first! */
int tx(port,ch)
    int port,ch;{
    outp(port,ch);
    return(ch);
    }

/* Receive a character */
int rx(port)
    int port;{
    return((int)inp(port));
    }

/* Wait for transmitter ready, then send character */
int musttx(port,ch,p)
    int port,ch;
    struct image *p;
    {
    unsigned t;
    int count;
    for(t=TIMEOUT; t!=0; t--)
    {
```

```
    if(txrdy(port))
       {
       tx(port,ch);
       p->charcount++;
       if (p->charcount>298)
         {
      printf("\nWaiting\n");
         while(mustrx(port)!=37);
      printf("Percent recieved\n");
         tx(COM1,' ');
         printf("Space sent\n");
         p->charcount = 0;
         }
       return(ch);
       }
    }
    return(0);
    }

/* Wait for character from port */
int mustrx(port)
    int port;
    {
    unsigned t;
    for(t=TIMEOUT; t!=0; t--)
       if(rxrdy(port)) return(rx(port));
    return(0);
    }
```

D.13 Data file for functions

```
INIT
0
#FRM
0
FRM-SZ
0
#CAM
1
LIVE
0
SNAP
1
ON
1
OFF
0
#P-MAP
0
P-MAP#
1
R-P-MAP
2
W-P-MAP
-1
#O-MAP
0
O-MAP#
1
C-MAP#
1
R-O-MAP
2
W-O-MAP
-1
MOV
10
TRANS
5
COPY
10
SUB
15
ADD
15
MULT
15
#COEF
0
COEF#
3
```

```
R-COEF
2
W-COEF
-1
CONV
10
R-VAL
2
HIST
5
PRO-X
5
PRO-Y
5
QUIT
0
TEST
0
0
0
```