VIP Version 4.0
User's Manual

Peter Kovesi, Carl Fisher and Du Huynh

Robotics and Vision Research Group
Department of Computer Science
The University of Western Australia

March 1994

VIP Version 4.0 Users Manual

Robotics and Vision Research Group
Department of Computer Science
The University of Western Australia
Nedlands, W.A. 6009.

Copyright

VIP is available via ftp from

cs.uwa.edu.au

The software can be found in /pub/robvis/vip

Acknowledgement

The Robotics and Vision Research Group acknowledge the support received from Digital through their External Research Programme.

Contents

Chapter 1

Introduction

VIP (Visual Information Processing) is an image processing software library that has been developed by the Robotics and Vision Research Group in The Department of Computer Science at The University of Western Australia. The system consists of two parts:

1. A library of C functions for manipulating and transforming images that can be incorporated in your own programs.
2. A set of stand-alone programs for performing many of the more commonly used operations on images.

The VIP library of image processing functions has been designed to enable users to create their own image processing programs very quickly and simply using a toolkit of standardized library functions.

The library includes functions in the following categories:

- image input/output and memory management
- image type and data format conversion
- image enhancement and spatial transformations
- mask operations
- arithmetic operations
- image statistics
- frequency domain transformations
- camera calibration and 3D data reconstruction
- vector and matrix manipulation
- one dimensional edge detection
- complex arithmetic
- miscellaneous functions

The VIP image data structure supports a wide variety of image types ranging from a basic 8 bit grey level representation through to complex valued and colour images. New image types can be easily added without 'breaking' the existing software. An additional feature of the VIP image data structure that distinguishes it from other image structures is its provision for comprehensive camera calibration data and lens distortion parameters. The VIP system provides support for reconstruction of 3D information from images via structured light and stereo.

The VIP System is designed to run under Unix on DEC Alpha, Sun Sparc Machines and PC's running LINUX, a subset of VIP is also available to run under DOS.

1.1 History

The origins of the system go back 1986 when The Computer Science Department at The University of Western Australia started to develop a library of image processing routines, coining the name 'VIP'. At about the same time the Automated Sheep Shearing Group in The Department of Mechanical Engineering started to develop a vision system for the automated measurement of sheep. Each group devised their own, different image data structures, which required conversion when they were exchanged between groups. In 1991, with increasing collaboration between the two groups, and with each wishing to change and enhance the image data structures they were using, a new common structure was devised. The two software libraries were merged and considerably enhanced. However, shortly after this, vision research within the Automated Sheep Shearing Group ceased due to changes in research directions. Almost all of the subsequent development has been done within The Department of Computer Science.

1.2 Authors

A number of people have contributed to the system. The initial work on VIP in the Computer Science Department was done by Chris Avis. The Automated Sheep Shearing Group's vision system was developed by Peter Kovesi. Much of the subsequent development in the Department of Computer Science has been done by Du Huynh and Peter Kovesi. Recently Carl Fisher standardized the code to ANSI C, placed the system under SCCS, cleaned up much of the code and fixed numerous bugs. He also wrote the Motif version of the image displaying program, splm. Contributions to the system have also been made by Graham Walker, James Trevelyan, James Cooper, Ian Reid, Svetha Venkatesh, Ben Robbins and Mike Robins.

1.3 Bug Reporting

Please report any bugs to:

`pk@cs.uwa.edu.au`

or post them to the newsgroup

`cs.vip`

Chapter 2

Getting Started With VIP

This section describes how to set up your environment variables in order to give you access to VIP programs, libraries and on-line manuals. Some examples are also provided to help you avoid having to read this manual in too much detail!

2.1 Setting environment variables

To gain access to the executable VIP programs add the following to your path specification in your .cshrc file

```
setenv PATH  "... your existing path...  :/proj/robvis/vip/bin/sunbin"
```

Note: If you are working on the DEC alpha you should specify 'decbin' rather than 'sunbin'.

To gain access to the manual entries you need to modify your MANPATH setting

```
setenv MANPATH "... your existing path...:/proj/robvis/vip/man"
```

in your .cshrc or .login file.

2.2 Image formats

VIP has its own image format, what distinguishes this format from others is that it has provision for storing camera calibration and lens distortion parameters in the image header. If you obtain images using the programs SNAP using the Videopix on the Sun or PKSNAP on the PC in the RobVis lab they will be stored in the VIP format. VIP image files generally have a .im ending.

Other common image formats that you may have to deal with are GIF, TIFF, PPM and PGM. Conversion utilities exist to convert images between most of these formats.

2.2.1 Image Format Conversion

To convert an image from VIP to PPM (Portable PixMap) use:

```
% vip2ppm  [vip_image_file] [-o ppm_image_file]
```

To convert a PPM, PGM or PBM image to VIP format use:

```
% pm2vip [pm image_file] [-o vip_image_file]
```

To convert from PPM to PGM (Portable Grey Map) use:

```
% ppmtopgm PPM_image_name > PGM_image_name
```

To convert from TIFF to VIP use:

```
%tifftopnm TIFF_image_name > dump
%pm2vip dump -o VIP_image_name
```

To convert a VIP image to or from the Sun raster format use:

```
% vip2raster [vip_image_file] [-o raster_image_file]

% raster2vip [raster_image_file] [-o vip_image_file]
```

To work with other image formats the general rule is to convert to PPM or PGM and then convert from there to your destination format. Do a 'man -k ppm' to find all the conversion utilities that work with PPM.

2.3 How to Display Images

There are a number of image displaying programs: splash, splm, xdisplay, xv and surface. xdisplay and xv are the ones that you will find most useful on the monochrome X terminals as they are capable of providing a dithered display. You can find some sample images to display in /proj/robvis/vip/images.

```
% xdisplay vipimage_name
```

The xdisplay program converts your VIP image into ppm format and uses the public domain xv utility to display your image. Click on the right mouse button with the cursor in the image to get a control panel. The '4x3' button will give you a more natural display as camera pixels have an aspect ratio of about 4x3 but the display screen has square pixels. Clicking on the 'coledit' button will bring up another control panel. Most of the controls relate to manipulation of the colour map. For grey scale images the intensity control is of most interest. This panel allows you to graphically manipulate the mapping from input grey levels to the displayed grey levels. By manipulating the mapping you can negate the image, threshold it, suppress detail in some sections of the grey scale and enhance it in others.

```
% xv [image_name]
```

xv is a public domain X11 program that displays images in the GIF, PBM, PGM, PPM, TIFF, X11 bitmap, JPEG, Sun Rasterfile, and PM formats on 1-, 4-, 6-, 8-, 16-, 24-, and 32-bit X displays. xv will also read compressed versions of these files. See the manual entry for more information.

```
% splash [vipimage_name]
```

Splash only displays VIP images and only runs on colour displays. Image display is quite fast, 1D slices through the image can be digitised, the lookup table can be interactively adjusted, plus a number of other operations are available.

```
% splm [vipimage_name]
```

Splm is a Motif version of splash. It also allows interactive thresholding of images.

```
% surface <image_name>
```

Surface displays VIP images as a 3D surface with interactive control over viewing direction. It can be rather slow when dealing with large images.

See manual entries for more details.

2.4 Some VIP programs to try

To negate an image use:

```
% negim image_name -o output_image
```

To threshold an image (make it binary black or white) use:

```
% thresh image_name -t threshold_value -o output_image
```

To obtain a histogram of the occurrence of each intensity value in an image use:

```
% histogram image_name -o outputfile
```

This produces a data file of values. You can display this using gnuplot.

```
% gnuplot
gnuplot> plot "datafile" with lines
```

gnuplot is a very powerful plotting utility, it is suggested that you become familiar with it.

The histogram plot can be useful in selecting a suitable threshold when generating a binary image.

To histogram equalize (make the histogram of intensity values more evenly spread, aiding interpretation) an image use:

```
% histe image_name -o output_image
```

2.5 Obtaining hard copy of images

To obtain a hard copy of your image, use the program, laser. This program converts an image in VIP format to postscript. The output may be piped directly to a laser printer or saved in a file for subsequent inclusion in a document.

For example, to obtain a hard copy of an image, sized 16cm x 12cm (providing a 4 x 3 aspect ratio of a 'square' image), placed in the centre of the page, and sending the output to the printer 'sp', use:

```
% laser image_name -w 16 12 -p ce | lpr -Psp
```

See the manual entry for more information.

2.6 Writing Programs and Functions that call VIP Libraries

Writing programs and functions that use VIP library functions is relatively straightforward. Indeed the aim of VIP has been to make the construction of image processing functions and programs as quick and simple as possible. Programs, even image processing ones, generally spend far more time being written and debugged than they do actually running.

A convention that has been adopted in VIP is that all image processing functions do not destroy or overwrite their input image(s). Memory for a new image is always allocated and the result placed there. The address of the new image is usually returned by the function. While this can have some penalty in speed and certainly in memory usage it results in programs having a simple and consistant structure.

Adopting this convention simplifies cases where one wants to mix images of different types. Also, in many cases one does not want the type of the output image to be the same as that of the input image, for example, the square of a BYTETYPE image will produce values outside of the range 0-255, it makes sense to return an integer or floating point valued image. Similarily when convolving an image with a first or second derivative mask the results will contain negative values and again one would want to return a signed integer or floating point valued image.

To minimize the time spent tracking down bugs VIP incorporates some error checking. One of the most common problems is that illegal memory access where one mistakenly tries to access locations outside the image bounds. VIP images incorporate memory sentinel values at the start and the end of the image data. A function, Image_OK, can be called by users to check the validity of image data at any time. This function checks that the image pointer is not NULL, checks that it has the appropriate idcode for an image and checks that the memory sentinel values have not been corrupted (indicating one has written data beyond the bounds of the image). This function is called by all VIP functions prior to any processing of images to ensure errors of this sort are trapped as soon as possible.

Three example files are provided on the following pages:

1. A program that reads an image of BYTETYPE and thresholds it. Values in the input image that are below the threshold are set to zero (black), values greater than the threshold are set to 255 (white). The resulting image is then written to a file.

2. A sample makefile for programs that use VIP.

3. A function that performs a homomorphic image enhancing filtering operation on an image. This example illustrates the use of a number of different image types.

To gain access to the VIP functions one simply uses the following include statement in one's source file.

```
#include "vip.h"
```

2.6.1 A Sample Program

```
/** threshold.c ------------------------------------------------

Sample program that illustrates use of the VIP library.

This program thresholds an image.  Values in the input image that are
below the threshold are set to zero (black), values greater than the
threshold are set to 255 (white).

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

#include <stdio.h>
#include "vip.h"                /* the VIP include file */

void main(int argc, char *argv[])
{
    IMAGE  *inimage = NULL, *outimage = NULL; /* pointers to the images we will use */
    int     rows, cols, row, col;
    int     threshold;

    if (argc != 4) {
        printf("usage: vipexample in_image threshold  out_image \n");
        return;
    }

/* Read the image into memory.  */
    inimage = Read_Image(argv[1]);

/* Check that we read the data from the file.  Note that if Read_Image
   was unable to read the image it would have produced an error message
   via the function VIP_Error_Msg so the following message is really
   not needed. */
    if (!inimage) {
        printf("Could not open %s\n", argv[1]);
        return;
    }

/* Check that it is a BYTETYPE image */
    if(inimage->type != BYTETYPE) {
        printf("Threshold can only operate on BYTETYPE images\n");
        return;
    }

    threshold = atoi(argv[2]);

    if(threshold < 0 || threshold > 255) {
        printf("threshold value must be in range 0 - 255\n");
Free_Image(inimage);
        return;
    }

    rows = inimage->rows;
    cols = inimage->cols;

/* Allocate space for the output image */
    outimage = Allocate_Image(0, 0, rows, cols, BYTETYPE);

    if (!outimage) {
```

```
        printf("Could not allocate space for output image\n");
        Free_Image(inimage);
        return;
    }

/*
     Now do the thresholding
     Note that there is a VIP library function to do this
     and the code below could simply be replaced with:
     outimage = Thresh_Image(inimage, threshold);
*/

    for (row = 0; row < rows; row++) {
        for (col = 0; col < cols; col++) {
    if (inimage->i.c[row][col] > threshold)
                outimage->i.c[row][col] = 255;
    else
                outimage->i.c[row][col] = 0;
        }
    }

/* Write the image to the file name specified by argv[3] */
    Write_Image(outimage, argv[3]);

/* Free the memory allocated for images */
    Free_Image(inimage);
    Free_Image(outimage);

    return;
}
```

Note that in this program individual pixels in the input image, inimage, which is of BYTETYPE, are referenced using

```
inimage->i.c[row][col]
```

The pointer, i, is a union structure and can be treated as a pointer to any of the currently supported image data types.

For other types of images one has to specify how the union structure, i, should be used. For example to access the element in the eleventh row and ninth column in a LONGTYPE image one would use:

```
value = image->i.l[10][8];
```

In a FLOATTYPE image one would address pixels using

```
value = image->i.f[10][8];
```

If the image is of RGBTYPE one would access the colour components using:

```
redvalue   = image->i.rgb[10][8][0];
greenvalue = image->i.rgb[10][8][1];
bluevalue  = image->i.rgb[10][8][2];
```

If the image is of COMPLEXTYPE one would use:

```
realvalue      = image->i.cx[10][8].r;
imaginaryvalue = image->i.cx[10][8].i;
```

See the section, "The VIP IMAGE Structure" for more details.

2.6.2 A Sample Makefile

Compiling and linking a program using VIP is illustrated by the example makefile listed below.

Note that you should use the gcc compiler as the VIP system is written in ANSI C. (use 'setenv CC gcc' in your .cshrc file).

```
# Example makefile using VIP

# First set up directories and pathways.

VIP_DIR = /proj/robvis/vip
LIB_PATH = -L$(VIP_DIR)/lib/sunlib  # use declib on the Alpha
INCLUDE_PATH = -I$(VIP_DIR)/include

# Compiling flags
CC = gcc

fla = -DSUN                     # use -DDEC_ALPHA on the Alpha
CFLAGS = $(INCLUDE_PATH) $(fla)

# Specify the libraries - we want the VIP and maths libraries
LIBS = $(LIB_PATH) -lvip -lm

# Now make the program

yourprog: yourprog.o
	$(LINK.c) -o yourprog yourprog.o $(LIBS)

yourprog.o: yourprog.c
	$(COMPILE.c) yourprog.c
```

2.6.3 A Sample Function

The following function performs a homomorphic filtering operation on an image. This is an image enhancing process that attempts to reduce the effects of low frequency lighting variations across an image, yet at the same time accentuating high frequency intensity variations to highlight the detail (see Gonzalez and Woods *"Digital Image Processing"* p. 213 for more detail). The steps involved in this process are:

1. The log of the image is taken.
2. The Fast Fourier Transform is taken.
3. A filter function is constructed that reduces low frequency components in the image and magnifies the high frequency ones.
4. The Fourier Transform of the image and the filter are multiplied together.
5. The inverse Fourier Transform is taken.
6. The exponential of the image is taken.
7. A percentage of the extremes of the image histogram are truncated.
8. Finally the image values are rescaled to the range 0-255.

You should note the following:

1. Different image types are accepted and, on some occasions, mixed by the VIP functions that are called. Log_Image is passed a BYTETYPE image and returns a FLOATTYPE. FFT_Image is passed a FLOATTYPE image and returns a COMPLEXTYPE. Multiply_Image is passed a COMPLEXTYPE and a FLOATTYPE image and returns a COMPLEXTYPE. This ability to cope with different image types simplifies the construction of programs.
2. There is no checking between calls to see if the previous function call failed. In general VIP functions check to see if an input image is valid (non-NULL pointer and correct ID code). If an input image is not valid the function will report the error and exit returning a NULL image pointer. Thus if an error occurs early in the sequence one will simply cascade harmlessly (hopefully) through the other function calls before terminating. Not having to embed large numbers of error trapping calls in one's code makes it much clearer to read.
3. All image pointers are initialised to NULL. This is important defensive programming. If any image pointer is passed to a function, and that pointer has never had any image data assigned to it, the receiving function can detect this through its value being NULL. Also, the function Free_Image does not attempt to free memory if it is passed a NULL pointer. This is important if one is to cascade safely through a sequence of function calls should an error occur.

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

Function to perform homomorphic filtering on an image. An image
enhancing process that attempts to reduce the effects of low frequency
lighting variations across an image, yet at the same time accentuating
high frequency intensity variations to highlight the detail.
Reference: Gonzalez and Woods "Digital Image Processing" page 213.

Peter Kovesi  January 1994

input: image    - input image.
       LowGain  - gain factor for low frequencies (suggest 0.5).
       HighGain - gain factor for high frequencies (suggest 2.0).
       CutOff   - frequency at which transition from LowGain to
                  HighGain occurs (suggest 0.4).
       order    - order of Butterworth type filter (suggest 1)
       lHistogramCut - percentage of low end of histogram to
                       truncate (suggest 1).
       uHistogramCut - percentage of high end of histogram to
                       truncate (suggest 5).

Function returns filtered image.

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


IMAGE  *Homomorphic(IMAGE * image, double LowGain, double HighGain,
            double CutOff, int order, int lHistogramCut, int uHistogramCut)
{
    IMAGE  *t1 = NULL, *t2 = NULL, *t3 = NULL, *t4 = NULL, *t5 = NULL,
           *t6 = NULL, *t7 = NULL;
    IMAGE  *outimage = NULL, *filter = NULL;
    int     r, c;


    t1 = Log_Image(image);

    t2 = FFT_Image(t1, 0);
    Free_Image(t1);

    filter = Make_Homomorphic_Filter(t2->rows, t2->cols, CutOff, LowGain,
HighGain, order);

    t3 = Multiply_Image(t2, filter);
    Free_Image(t2);
    Free_Image(filter);

    t4 = IFFT_Image(t3, 0);
    Free_Image(t3);
    if(!t4) {
      VIP_Error_Msg("Homomorphic: An error occurred during a function call");
      return(NULL);
    }


/* Extract real part of image */

    t5 = Allocate_Image(0, 0, t4->rows, t4->cols, FLOATTYPE);
```

```
    for (r = 0; r < t4->rows; r++)
        for (c = 0; c < t4->cols; c++)
            t5->i.f[r][c] = t4->i.cx[r][c].r;

    Free_Image(t4);

    t6 = Exp_Image(t5);
    Free_Image(t5);

/* Remove outlying values of image by truncating top and lower ends of
the image histogram by HistogramCut percent */

    t7 = Truncate_Image_Histogram(t6, lHistogramCut, uHistogramCut);
    Free_Image(t6);

    outimage = Float2Byte_Image_X(t7);
    Free_Image(t7);

    if(!outimage)
      VIP_Error_Msg(``Homomorphic: An error occurred during a function call'');

    return (outimage);
}
```

Chapter 3

The VIP System in More Detail

3.1 The VIP IMAGE Structure

Central to any image processing system is the image data structure used. The VIP IMAGE structure supports a wide variety of image types ranging from a basic 8 bit grey level representation through to complex valued and colour images. This is achieved by using a union structure to point to the actual image data. This union structure consists of pointers to all the different types of image data elements that are supported. As pointers to all data types are the same size any number of new pointers to new data types can be added to the image pointer union without affecting the existing system. Thus new image types can be added easily without 'breaking' the system. When accessing image data, simply choosing the appropriate element from the image pointer union will result in the correct pointer arithmetic occurring.

An additional feature of the VIP image data structure that distinguishes it from other image structures is its provision for comprehensive camera calibration data and lens distortion parameters. The VIP system provides support for reconstruction of 3D information from images via structured light and stereo.

To aid portability the data structures have been designed to have all elements aligned with word boundaries so that the different packing characteristics of different machines would not be an issue. It is assumed that binary files are stored in big-endian format (byte reversed order), this is the natural format produced by Sun SPARC machines. VIP libraries compiled for little-endian machines (DEC Alpha and PC's) will perform byte-reversal of image data when reading so that it can be interpreted correctly, and then when writing data out, will also perform byte-reversal to ensure consistency of binary data files. Unfortunately this does, of course, involve some loss of efficiency.

The image types currently available under VIP are:

BYTETYPE
An unsigned 8 bit grey level image with values ranging from 0 to 255.

SHORTTYPE
Signed 16 bit grey level image.

LONGTYPE
Signed 32 bit grey level image.

FLOATTYPE

32 bit floating point valued image.

DOUBLETYPE
64 bit floating point valued image.

COMPLEXTYPE
Complex valued image, real and imaginary components each represented by a 32 bit floating point value.

RGBTYPE
Colour image represented by red, green and blue components, each component represented by an 8 bit unsigned values.

HSITYPE
Colour image represented by hue, saturation and intensity components, each component represented by an 8 bit unsigned values.

Where appropriate (and where it makes sense) the VIP image processing libraries will in most cases allow different image types to be used and/or mixed in a seamless manner.

The VIP image data structure is listed below. For full details refer to the listing vip.h in the vip/include directory.

```
/*- Image types ------------------------------------------------------*/

#define BYTETYPE    1
#define SHORTTYPE   2
#define LONGTYPE    3
#define FLOATTYPE   4
#define DOUBLETYPE  5
#define COMPLEXTYPE 6
#define RGBTYPE     7
#define HSITYPE     8


/*-  Data types used to build IMAGE structure  ---------------------*/

typedef struct { /* complex number */
    float    r;
    float    i;
}        COMPLEX;

typedef unsigned char RGB[4]; /* RGB colour image, red green and blue
 * stored in an array of 4 bytes for
 * convenience and efficiency */

typedef unsigned char HSI[4]; /* Hue, saturation and intensity colour
                                 * image */

typedef unsigned char *c_array;
typedef short *s_array;
typedef long *l_array;
typedef float *f_array;
typedef double *d_array;
typedef COMPLEX *cx_array;
typedef RGB *rgb_array;
typedef HSI *hsi_array;
```

```
typedef union imagedata {
    c_array *c;
    s_array *s;
    l_array *l;
    f_array *f;
    d_array *d;
    cx_array *cx;
    rgb_array *rgb;
    hsi_array *hsi;
}         imagedata;



/*-  CAMERA structure  -------------------------------------------*/

typedef struct {
    int      idcode;
    char     version[8];
    int      channel;           /* Channel on PIP board */
    int      gain, offset;      /* Gain and offset for the channel */
    int      uvoffset;          /* UV offset in using calibration matrix
                                 * and lens compensation data */
    int      calibstatus;       /* Flag indicating type of calibration data
                                 * present */
    double   gamma[5];          /* For grey scale correction if needed */
    double   calib_matrix[3][4];/* Calibration matrix */
    double   Ucomp[18];         /* Coeffs of lens compensation polynomial*/
    double   Vcomp[18];         /* Coeffs of lens compensation polynomial*/
}         CAMERA;


/*- IMAGE Data Structure -------------------------------------------*/

typedef struct {
    int      idcode;            /* Code to allow automatic identification*/
    char     version[8];        /* software version No */
    int      type;              /* Data type - char, rgb colour etc */
    int      umin;              /* Image origin coords */
    int      vmin;
    int      rows;
    int      cols;
    double   resolution;
    char     info[80];          /* Space for comments */
    CAMERA   camera;
    long     reserved[3];
    void    *pntr[2];           /* two pointers for attaching other
                                 * structures to the image, or for linking
                                 * sequences of images together */
    union imagedata i;
}         IMAGE;
```

The following table lists how pixel elements in different image types are accessed.

```
value = image->i.c[row][col];   /* BYTETYPE */
value = image->i.s[row][col];   /* SHORTTYPE */
```

```
value = image->i.l[row][col];   /* LONGTYPE */
value = image->i.f[row][col];   /* FLOATTYPE */
value = image->i.d[row][col];   /* DOUBLETYPE */

RedValue   = image->i.rgb[row][col][0];  /* RGBTYPE */
GreenValue = image->i.rgb[row][col][1];
BlueValue  = image->i.rgb[row][col][2];

HueValue        = image->i.hsi[row][col][0];  /* HSITYPE */
SaturationValue = image->i.hsi[row][col][1];
IntensityValue  = image->i.hsi[row][col][2];

RealValue      = image->i.cx[10][8].r; /* COMPLEXTYPE */
ImaginaryValue = image->i.cx[10][8].i;
```

3.2 Global Variables in VIP

There are a number of global variables used in the VIP system. They are declared in vipglobals.c which is listed below.

```
FILE    *VIP_LOG_FILE = NULL;  /* Pointer to log file for data recording*/
FILE    *VIP_STD_ERR = stderr; /* Error output */
int      VIP_VERBOSE = 0;      /* flag to turn on or off extra printing of
                                * data */
int      VIP_DISPLAY = 0;      /* flag for when one wants to turn
                                * displaying of images on or off */
char     VIP_MESSAGE[VIP_MESSAGE_LENGTH];      /* global message variable
                                                * for passing to X view
                                                * alert boxes etc */

double   PLANE[USERPLANEMAX][3][3];     /* Fifty constraint planes, each
                                         * defined by 3 points. These are
                                         * used for 3D light stripe
                                         * reconstruction */
```

3.3 Conventions

VIP function names start with capitals, multiple words within a function name are separated by underscores and each word starts with a capital, eg. Allocate_Image.

VIP data types and VIP global variables are in capitals.

All error messages are reported via VIP_Error_Msg (in vipio.c).

The success or otherwise of the execution of a function is indicated by the return of either 'OK' (1) or 'ERROR' (0). Functions that return addresses return NULL on failure.

In general VIP functions should check to see if an input image is valid (non-NULL pointer and correct ID code). If an input image is not valid the function will report the error and exit returning a NULL image pointer. Thus if an error occurs early in a sequence of function calls in a program one will simply cascade harmlessly through the other function calls before terminating. The idea here being to avoid having to embed large numbers of error trapping calls in one's code, thus making it much easier to read.

Accordingly it is good practice to initialise all image pointers to NULL. If any image pointer is passed to a function, and that pointer has never had any image data assigned to it, the receiving function can detect this as its value will be NULL. Also, the function Free_Image does not attempt to free memory if it is passed a NULL pointer. This is important if one is to cascade safely through a sequence of function calls if an error occurs as described above.

Any image processing function should not destroy or overwrite its input image(s). The result should be placed in a new image, with the address of the new image usually being returned by the function.

Generally, each image processing function contains a series of CASE statements, indexed by the image type, containing code to process the different types of images available under VIP. For example a thresholding function could operate on BYTETYPE, SHORTTYPE, FLOATTYPE and other images using this mechanism. There is no requirement that functions cater for all image types, and in many cases they will only support BYTETYPE images. Each function will test for validity of the image type and return an error message if it does not support the particular image input type. It is assumed that as people find functions that do not support an image type that they need they will add the appropriate code to these functions (and update the documentation!).

3.4 Error Reporting

A flexible system exists for the reporting of errors. Different types of programs have different needs for error reporting. Command line programs simply want messages printed on the screen but windows based programs want messages to be displayed in alert boxes. Both of these needs can be accommodated in the VIP system. As mentioned above the success or otherwise of the execution of a function is indicated by the return of either 'OK' (1) or 'ERROR' (0). Functions that return addresses return NULL on failure. All error messages are reported via the function VIP_Error_Msg (in vipio.c). This function allows error messages to be directed to up to three different locations depending on the needs of the user.

The three different (globally accessible) locations are:

VIP_STD_ERR
The default value of this is stderr, but it can be assigned to any file pointer.

VIP_LOG_FILE
A file pointer, default value is NULL.

VIP_MESSAGE
A character array. Error messages are always copied into this array (space permitting).

For standard command line based programs the default configuration simply results in error messages being sent to stderr (your screen).

For an X View or other windows based program it is not appropriate to be sending text messages to consoles. In these situations VIP_STD_ERR should be set to NULL to suppress the printing of messages and the array VIP_MESSAGE used instead. When an error is detected through a zero or NULL value returned from a function the current contents of VIP_MESSAGE can be passed to an alert box for display. Multiple error messages are concatenated together so that one can trace the source of any errors. Once the message has been displayed in an alert box it is up to the programmer to ensure that the VIP_MESSAGE array is cleared to allow new errors to be reported to it.

The VIP_LOG_FILE pointer is intended for programs that run continuously for an extended period of time, say, on a long sequence of images. In these situations one may wish errors to be recorded silently in a log file for later analysis if needed.

3.5 Location of the VIP System Components

VIP Library :

DEC Alpha UNIX compatible VIP library : /proj/robvis/vip/lib/declib/libvip.a
SUN Sparc UNIX compatible VIP library : /proj/robvis/vip/lib/sunlib/libvip.a
PC DOS compatible VIP library : /proj/robvis/vip/lib/pclib/libvip.a

VIP Utilities :

DEC Alpha UNIX compatible VIP utilities : /proj/robvis/vip/bin/decbin
SUN Sparc UNIX compatible VIP library : /proj/robvis/vip/bin/sunbin

VIP Source Code :

The source code for the various hardware and software platforms on which the VIP System is implemented, is represented by a single set of code.

VIP Library Code : /proj/robvis/vip/src/lib
VIP Utilities Code : /proj/robvis/vip/src/vip
VIP Include Files : /proj/robvis/vip/include

There are additional subsets of VIP related code, designed for particular VIP related tasks; these are discussed in the appendix.

VIP Manuals :

VIP Utilities : /proj/robvis/vip/man/man1
VIP Library : /proj/robvis/vip/man/man2

The LaTeX source files for this manual can be found in /proj/robvis/man/mantex. All the on-line manuals are created automatically from these sources.

3.6 Files

vip.h

The VIP include file.

vipglobalsfn.h
vipglobals.c

File declaring all global variables used in the VIP system.

vipiofn.h
vipio.c

General input/output functions, image allocation and freeing, conversion between different formats.

vipspatfn.h
vipspat.c

Collection of basic spatial image processing functions.

vip1Dfn.h
vip1D.c

Collection of functions for processing one dimensional slices through images.

vipfreqfn.h
vipfreq.c

Collection of frequency based image processing functions.

vipcomplexfn.h
vipcomplex.c

Functions for manipulating complex numbers.

vipcalibfn.h
vipcalib.c

Functions supporting 2D/3D image transformations, reconstruction of 3D data via structured light and stereo.

vector.h
vectorfn.h
vector.c

An extensive library of vector and matrix manipulation routines.

Chapter 4

VIP Programs

4.1 Image Display Programs

splash	-	image displaying program.
splm	-	vip image displaying program, using Motif. (Currently on DEC Alpha only).
xdisplay	-	display a VIP image using xv.
laser	-	laser printer filter
surface	-	display a VIP image as a 2D surface.

4.2 Image Type and Format Conversion

pm2vip	-	convert a PPM, PGM or PBM image to VIP format.
vip2ppm	-	convert a VIP image to PPM format.
raster2vip	-	convert a Sun raster image to the VIP format.
vip2raster	-	convert a VIP image to the Sun raster format.
convert2byte	-	convert any VIP image to BYTETYPE.
convert2float	-	convert any VIP image to FLOATTYPE.
convert2cx	-	convert any VIP image to COMPLEXTYPE.
raw2vip	-	convert image from raw to VIP format.
vip2raw	-	strip off VIP image header to create a raw data file.
ap2cx	-	unite two files that contain amplitude and phase images to form a complex image.
ri2cx	-	merge the real and imaginary images to form a complex image.
cx2ri	-	separate the real and the imaginary components of a complex image.

4.3 Spatial Transformations

xyscale	-	rescale an image.
reflect	-	reflect an image about the row or column axes.
thresh	-	threshold image.
deres	-	deresolve image.
res	-	re-resolve/deresolve?.
snn	-	symmetric-nearest-neighbour smoothing an image.
knn	-	k-nearest-neighbour smoothing.
histe	-	perform histogram equalization on an image.
transpose	-	transpose an image.
clip	-	clip a VIP image to form a smaller image.

4.4 Arithmetic Operations

absim	-	absolute values of an image.
expim	-	output the inverse log (to base e) of an image.
logim	-	apply a logarithmic transformation to a floating point image.
negim	-	negate an image.
nlogim	-	apply the natral log operation to an image.
scaleim	-	pixelwise multiply an image by a scalar.
sqrim	-	pixelwise square an image.
sqrtim	-	pixelwise apply the square root operation to an image.
sumim	-	sum two matrices.

4.5 Edge Detection

3edge	-	find edges in image using local energy edge detector.
convolve	-	find edges in image using local energy edge detector.
canny	-	(documentation needed).
hyster	-	.
outedges	-	.
zc	-	label zero crossings in an image.

4.6 Image Information/Statistics

differ	-	compares two BYTETYPE images
histogram	-	produce a histogram data file from an image for plotting.
imageinfo	-	display information about an image.
spy	-	view image data values.
gline	-	produce a row data file for xgraph or xvgr from image files.

4.7 Test Image Creation

gauss	-	gaussian.
d2gauss	-	produce the second derivative of a Gaussian mask.
lapgauss	-	create a laplacian of gaussian image.
genimage	-	generate an image from a text file or stdin.
eqnim	-	Synthesized image generation from a given equation.
gnoise	-	add gaussian noise to image.

4.8 Frequency Based Image Operations

fft	-	fast fourier transform of image.
ifft	-	Inverse Fast Fourier Transform of image.
mag	-	calculate the amplitude of a complex image.
phase	-	calculate the phase angles of a complex image.
conj	-	return a pixelwise complex conjugate image.
lowpass	-	smoothing an image in the frequency domain using either ideal or Butterworth filters.
highpass	-	sharpening an image in the frequency domain using either ideal or Butterworth filters.
homomorphic	-	homomorphic image enhancement.

4.9 Others

calibrate	-	camera calibration.

Chapter 5

VIP Library Functions: Summary

The VIP Library

The VIP library is called libvip.a and its functions may be incorporated into user written programs. There is more than one VIP library of this name, each of these is stored in a place indicating the implementation platform (currently Sun, DEC Alpha and DOS, see the section, Location of the VIP System Components). The set of VIP library functions may be subdivided into several general classes :

- Input/Output and Memory Management Functions
- Image Type and Data Format Conversion Functions
- Image Enhancement and Spatial Transformation Functions
- Mask Functions
- Arithmetic Functions
- Statistical Functions
- Frequency Domain Functions
- Camera Calibration Functions
- Vector and Matrix Functions
- One Dimensional Edge Detector Functions
- Complex Arithmetic Functions
- Miscellaneous Functions

5.1 Input/Output and Memory Management Functions

See on-line manual vipio.

Allocate_Header	-	Allocates memory for an image header.
Allocate_Image	-	Allocates memory for an image.
Copy_Image	-	Copies an image.
Free_Image	-	Frees memory allocated for an image.
Read_Header	-	Reads in only the header part of an image.
Write_Header	-	Writes an image header.
Read_Image	-	Reads an image from a file.
Write_Image	-	Writes an image to a file.
Copy_Header	-	Copies the header information from an image to another.
Print_Header	-	Prints out the header information of an image.
Write_VFFImage	-	Writes a VIP image into a VFF format file.
Read_PCImage	-	Reads a binary file written by a PC and converts it into VIP format.
VIP2PPM_Image	-	Writes a VIP image into a PPM format file.

5.2 Image Type and Data Format Conversion Functions

See on-line manual vipconv.

Convert2Byte_Image	-	Converts an image of SHORT, LONG, FLOAT or DOUBLE TYPE to a BYTETYPE image.
Convert2Double_Image	-	Converts an image of BYTE, SHORT, LONG, FLOAT or LONG TYPE to a DOUBLETYPE image.
Convert2Float_Image	-	Converts an image of BYTE, SHORT, LONG or DOUBLE TYPE to a FLOATTYPE image.
Convert2Short_Image	-	Converts an image of BYTE, LONG, FLOAT or DOUBLE TYPE to a SHORTTYPE image.
Convert2Long_Image	-	Converts an image of BYTE, SHORT, FLOAT or DOUBLE TYPE to a LONGTYPE image.
Convert2Complex_Image	-	Converts an image of BYTE, SHORT, LONG, FLOAT or DOUBLE TYPE to a COMPLEXTYPE image.
Complex2PhaseFloat_Image	-	Converts a COMPLEXTYPE image to a FLOAT type Phase image.
Complex2MagFloat_Image	-	Converts a COMPLEXTYPE image to a FLOAT type Magnitude image.
Construct_Complex_Image	-	COMPLEX TYPE image from two FLOAT TYPE images.
Extract_Real	-	Extracts real component of a COMPLEX image.
Extract_Imaginary	-	Extracts imaginary component of a COMPLEX image.

5.3 Spatial Transformation Functions

See on-line manual vipspat.

Bilinear_Interpolate	-	Perform bilinear interpolation of grey values in an image.
XYScale_Image	-	Scale an image in x and y directions.
Rotate_Image	-	Rotate an image.
Truncate_Image_Histogram	-	Modify an image so that extremes in histogram are truncated.
Thresh_Image	-	Thresholds an image.
Floor_Thresh_Image	-	Zeroes out pixels whose values are lower than a threshold value.
Ceiling_Thresh_Image	-	Zeroes out pixels whose values are higher than a threshold value.
Apply_Local_Energy	-	Apply the Local Energy feature detector to an image.
Convolve_Image	-	Performs 2D convolution with an integer convolution mask.
FConvolve_Image	-	Performs 2D convolution with a float convolution mask.
Point_Convolve	-	Convolve a FLOAT MASK with an image at a specified point.
DeRes_Image	-	Reduce the resolution of an image.
Res_Image	-	Increase the resolution of an image (actually just scale it up).
Knn_Image	-	K nearest neighbour smoothing of an image.
Snn_Image	-	Symmetric Nearest Neighbour Smoothing of an image.
Clip_Image	-	Returns a sub-image from a larger image.
Image_Row_Reflect	-	Returns an image that is a horizontal mirror reflection.
Image_Column_Reflect	-	Returns an image that is a vertical mirror reflection.
Set_Look_Up_Table	-	Sets up a lookup table for modifying intensity values.
Erode_Image	-	Morphological erosion.
Dilate_Image	-	Morphological dilation.
Gaussian_1D	-	Produces a one-dimensional Gaussian FLOATTYPE image.
Gaussian_2D	-	Produces a two-dimensional Gaussian FLOATTYPE image.
Laplacian_Gaussian_1D	-	Produces a one-dimensional Laplacian of a Gaussian FLOATTYPE image.
Laplacian_Gaussian_2D	-	Produces a two-dimensional Laplacian of a Gaussian FLOATTYPE image.
Distance_Transform	-	Calculate a distance transform for a binary image.

5.4 Mask Functions

See on-line manual vipspat.

Read_Int_Mask	-	Reads an INT_MASK file into memory.
Allocate_Int_Mask	-	Allocate space for an INT_MASK structure.
Free_Int_Mask	-	Free memory allocated for an INT_MASK structure.
Write_Int_Mask	-	Writes a INT_MASK structure to a file.
Allocate_Float_Mask	-	Function to allocate space for a FLOAT_MASK structure.
Free_Float_Mask	-	Frees memory allocated for a FLOAT_MASK structure.
Float_Mask2Image	-	Convert a Float_Mask Structure to a Float Image.
Scale_Float_Mask	-	Scale values in a FLOAT_MASK.
Rotate_Float_Mask	-	Generate rotated versions of a reference mask.
Read_Energy_Filter	-	Reads an odd and an even filter for the Local Energey feature detector.

5.5 Graphics Functions

See on-line manual vipspat.

Image_Point	-	Function to draw a point in an image.
Image_Line	-	Function to draw a line in an image.

5.6 Arithmetic Functions

See on-line manual viparith.

Set_Image	-	Sets an entire image to a constant value.
Average_Image	-	Take the average of two images.
Diff_Image	-	Take the absolute difference of two images.
Squared_Image_Difference	-	Calculate squared difference between two images.
Subtract_Image	-	Subtract two images.
Add_Image	-	Add two images.
Exp_Image	-	Take the natural exponent of an image.
Log_Image	-	Take the natural log of an image.
Logarithmic_Transform_Image	-	Returns the logarithmic transform of an image.
Offset_Image	-	Add an offset to each pixel in an image.
Scale_Image	-	Function to pixel-wise scale an image by a given factor.
Sqr_Image	-	Squares an image.
Sqrt_Image	-	Take the square root of an image.
Multiply_Image	-	Pixelwise multiply two images together.
Divide_Image	-	Pixelwise divide two images.
Accumulate_Add_Image	-	Accumulation addition of an image to another.

5.7 Statistical Functions

See on-line manual vipstat.

Histogram_Image	-	Generate histogram of grey levels in a BYTETYPE image.
Image_Min_Max	-	Calculates the lower and upper bounds of pixel values of an image.
Image_Mean	-	Calculates the mean pixel value in an image.
Image_Count	-	Generate histogram of grey levels in grey level and colour images.
Image_Variance	-	Calculates the variance of pixel values of an image.

5.8 Frequency Domain Functions

See on-line manual vipfreq.

FFT_Shuffle	-	Reorders a complex image for the computation of a FFT.
Shift_Image_Origin	-	Shift the origin of an image so that the top-left corner is positioned at the image centre.
Image_Complex_Conjugate	-	Reverse the sign of the imaginary component of a COMPLEX image.
FFT	-	Overwrites complex image with its unscaled Fourier Transform.
FFT_Image	-	Returns FFT of an image.
IFFT_Image	-	Returns inverse FFT of an image.
FT_Image	-	Returns forward or inverse FFT of an image.
fourn	-	General N dimensional FFT function.
Make_Homomorphic_Filter	-	Create a Butterworth-like Homomorphic filter.

5.9 Camera Calibration Functions

See on-line manual vipcalib.

xyz2uv	-	Converts xyz world coordinates to integer uv image coordinates.
xyz2uvNC	-	Similar to xyz2uv but output values are not clamped to 0-511.
uv2xyz	-	Converts image uv coords to world xyz coords using a sensor structure.
Read_Plane_List	-	Reads a plane list file.
Set_Plane	-	Specify a plane.
Calc_Sensor	-	Calculates a sensor matrix from camera calibration matrix and a plane.
Stereo_Point	-	Calculates world xyz position of a point given the uv coordinates from two cameras.
Lens_Comp_3	-	Use 3rd order polynomial to compensate for lens distortion.
Lens_Comp_4	-	Use 4th order polynomial to compensate for lens distortion.
Calibrate	-	Camera calibration function.
Read_Calib_Pts	-	Reads a calibration data file containing 3D coordinates of calibration points.
Read_Image_Pts	-	Reads a calibration data file containing image data points.
Write_Camera_Txt	-	Writes a CAMERA data structure into a text file.
Read_Image_Pts_From_Calib_File	-	Reads image uv coords from a text camera calibration data file.
Read_Camera	-	Reads a binary camera data file.

5.10 Vector and Matrix Functions

See on-line manual vipvector.

Max_i	-	Returns the largest element in an array of integers.
Max_l	-	Returns the largest element in an array of longs.
Max_f	-	Returns the largest element in an array of floats.
Max_d	-	Returns the largest element in an array of doubles.
Min_i	-	Returns the smallest element in an array.
Min_l		
Min_f		
Min_d		
Limit_Value_i	-	Clamps a value to limits.
Limit_Value_l		
Limit_Value_f		
Limit_Value_d		
Out_Of_Range_i	-	Tests whether an variable is out of limits.
Out_Of_Range_l		
Out_Of_Range_f		
Out_Of_Range_d		
Move_i	-	Move values from one array to another.
Move_f		
Move_d		
Movexi	-	Move values from one array to another, arrays must not overlap.
Movexf		
Movexd		
Add_i	-	Add two vectors.
Add_d		
Sub_i	-	Subtract two vectors.
Sub_d		
Piv_i	-	Vector pivot function.
Piv_d		
Scal_i	-	Vector scale function.
Scal_d		
Crosp_i	-	Cross product function.
Crosp_d		
Norm_f	-	Normalize a vector and return its original magnitude.
Norm_d		
Mag_f	-	Return the magnitude of a vector.
Mag_d		
Copy_Int_To_Double	-	Copy array of ints to doubles.
Copy_Double_To_Int	-	Copy array of of doubles to ints.

Allocate_Matrix	-	Allocate memory for a matrix.
Free_Matrix	-	Free a matrix.
Mul_i	-	Matrix multiplication.
Mul_d		
Transpose_i	-	Transpose a matrix.
Transpose_d		
Homog_Trans	-	Homogeneous Transformation Function.
Gauss	-	Gaussian Elimination Function.
Gauss_Cond	-	Gaussian Elimination Function, returns condition number.
Print_Matrix_d	-	Matrix Printing Function.

5.11 One Dimensional Edge Detector Functions

See on-line manual vip1D.

Edge_1D	-	One dimensional edge detection function.
MEdge_1D	-	Multiple line search for an edge.
Convolve_Array_i	-	Convolve a 1D array of ints with a mask.
SD_Gauss_Mask_1D	-	Set up 2nd derivative of a Gaussian mask.
D_Gauss_Mask_1D	-	Set up 1st derivative of a Gaussian mask.
Gauss_Mask_1D	-	Set up a Gaussian mask.
Find_1st_Peak	-	Find the first peak in an array of integers.
Find_Closest_Peak	-	Find the peak in an array of integers that is closest to a specified position in the array.
Select_Peaks	-	Select a number of peaks in an array of integers.

5.12 Complex Arithmetic Functions

See on-line manual vipcomplex.

Complex_Mult	-	Multiply two complex numbers.
Complex_Div	-	Divide two complex numbers.
Complex_Sub	-	Subtract two complex numbers.
Complex_Add	-	Add two complex numbers.
Complex_Magnitude	-	Returns the magnitude of a complex number.
Init_Complex_Vector	-	Initialize elements of a complex vector to zero.
Allocate_Complex_Vector	-	Allocate memory for a complex vector.
Free_Complex_Vector	-	Free memory allocated for a complex vector.

5.13 Miscellaneous Functions

See on-line manual vipmisc.

Image_OK	-	Tests validity of an image structure.
Common_Window	-	Determines the common area between two images.
Find_Spot	-	Find the centroid of a blob within a window in an image.
Pixel_In_Image	-	Tests whether pixel coordinates are within image bounds.
VIP_Error_Msg	-	Function to handle error messages.
Wind_Int	-	Calculate average intensity in a window.
Is_2_Power	-	Tests if a number is a power of 2.
Power_Of_2_Greater_Or_Equal	-	Returns power of 2 greater than or equal to argument.
Closest_Power_Of_2	-	Returns the closest power of 2 to the argument.

Chapter 6

Program Reference

6.1 ABSIM

NAME

absim - absolute values of an image

SYNOPSIS

absim [image_file] [-o output_file]

DESCRIPTION

Absim reads an image from stdin, or from image_file if the argument is used and outputs an image in which each pixel contains the absolute value of the corresponding pixel of the input image. The output image is sent to stdin or written to output_file if specified.

-o output file name.

DIAGNOSTICS

Absim works for all VIP image types.

SEE ALSO

negim(1)

AUTHOR

Chris Avis; Du Huynh, 1992.

6.2 AP2CX

NAME

ap2cx - unite two files that contain the amplitude and phase images to form a complex image

SYNOPSIS

ap2cx -a amplitude_file -p phase_file [-o output_file]

DESCRIPTION

Ap2cx reads two floating point images from phase_file and amplitude_file and returns a complex image file. If either one of these files is missing, error message will be generated and the routine terminates. The output complex image is written to complex_file if the argument is given, or piped to standard output otherwise.

-p input file name for the phase angle.

-a input file name for the amplitude.

-o output file name.

SEE ALSO

fft(1), ifft(1), mag(1), merge(1), phase(1), cx2ri(1)

AUTHOR

Du Huynh

6.3 CALIBRATE

NAME

calibrate - Program for camera calibration.

CALIBRATE

This program calculates a 3x4 camera calibration matrix plus (optionally) a third order lens distortion compensation polynomial function. The input to the program is a set of 3D coordinates of calibration points along with the set of 2D image positions corresponding to these 3D points.

The program produces a binary file of a CAMERA structure which contains all the calibration data. A text file containing the calibration results and some accuracy analysis is also generated.

The program prompts you for the names of the data files to use. You have a choice of reading the image coordinates from a image data point file as specified below or from a text file containing the results of a previous calibration run. This can be useful when one is wanting to experiment with different calibration procedures.

The equations that have to be solved in the calibration process are Interrupt to calculate the calibration parameters are printed out for your reference. Usually the condition No of the matrix used to solve for the 3x4 camera calibration matrix will be of the order of several million. This is very, very poor (an ideal condition No is one) and the numerical accuracy of the machine is being pushed to the limit. The situation for the calculation of the lens distortion parameters is several orders of magnitude worse. Despite these problems the results seem to be OK and quite usable. However, you have been warned!

At least six points are required to calculate the calibration parameters. To obtain reasonable lens distortion compensation parameters the calibration points must be widely spread throughout the image. Using a large number of calibration points gives you greater immunity to noise and measurement error but paradoxically tends to make the matrices even more poorly conditioned. Reasonable results have been obtained using around 20-25 points.

DATA FILE FORMATS

The expected format of the data file containing the 3D coordinates of the calibration points is as follows:

```
File description, comments etc

point 1 x y z
point 2 x y z
  .   . . . .
point n x y z
```

All lines in the file are ignored unless they start with the word 'POINT'. Hence, text description can be used liberally. Point numbers are ignored and are simply there for convenient reference.

The expected format of the file containing the corresponding images coordinates of the calibration points is similar.

```
File description, comments etc

imagepoint 1 u v
imagepoint 2 u v
  .   . . . .
imagepoint n u v
```

All lines in the file are ignored unless they start with the word 'IMAGEPOINT'. Hence, text description can be used liberally. Point numbers are ignored and are simply there for convenient reference.

SAMPLE OUTPUT

```
* Calibration Data:   dump.txt

channel      0
gain         0
offset       0
uvoffset     256
calib\_status 2

Calibration Data
  -299.610620785     -2.055607208      2.822090003   -255.335663910
    -4.454247296    415.270368141    -52.312128177     -8.370913904
    -0.024475187     -0.063408615     -0.533921014      1.000000000


Lens Distortion Parameters

           Ucomp                        Vcomp

    7.92564267353643e-01      1.58476229228871e+00
    9.79488953699619e-01      5.13534034081102e-04
   -1.12084742552119e-02      1.02568871297902e+00
   -1.65402116510121e-05     -4.04312130036963e-05
    7.28040592258594e-06      9.28192201824597e-06
   -1.98441107827432e-05     -3.04409490435621e-05
    5.66944838827446e-07     -1.25982036041730e-07
   -5.62542972245705e-08      3.39363339712683e-07
    1.21502070349020e-07      1.66049145490563e-07
    4.35831088365635e-07     -1.12097189769947e-06
    0.00000000000000e+00      0.00000000000000e+00
    0.00000000000000e+00      0.00000000000000e+00
    0.00000000000000e+00      0.00000000000000e+00
    0.00000000000000e+00      0.00000000000000e+00
    0.00000000000000e+00      0.00000000000000e+00
    0.00000000000000e+00      0.00000000000000e+00
    0.00000000000000e+00      0.00000000000000e+00
```

```
   0.00000000000000e+00      0.00000000000000e+00


 Calibration Accuracy Data
1st Col is observed image position
2nd Col is corrected image position
3rd Col is ideal image position
4th Col is difference between observed and ideal image positions
5th Col is difference between compensated and ideal image positions
  1     59.000     57.522     58.378    0.62160   -0.85673
        75.000     73.753     73.847    1.15326   -0.09346
  2    221.000    221.219    221.509   -0.50861   -0.28943
        76.000     78.245     78.622   -2.62226   -0.37700
  3    347.000    345.538    345.459    1.54059    0.07908
        81.000     82.605     82.251   -1.25083    0.35399
  4    450.000    450.347    449.728    0.27220    0.61872
        87.000     84.900     85.303    1.69678   -0.40322
  5    420.000    419.745    420.616   -0.61613   -0.87144
       106.000    104.286    103.756    2.24400    0.53016
  6    332.000    331.187    331.561    0.43918   -0.37366
       100.000    100.734    101.291   -1.29132   -0.55766
  7    226.000    226.886    225.989    0.01145    0.89717
        98.000     99.010     98.370   -0.36953    0.64094
  8     88.000     88.511     87.530    0.46991    0.98062
        96.000     94.660     94.538    1.46241    0.12199
  9     83.000     84.087     82.857    0.14345    1.23002
       243.000    243.486    243.142   -0.14204    0.34346
 10    222.000    223.572    224.553   -2.55273   -0.98095
       245.000    246.239    245.366   -0.36599    0.87323
 11    334.000    333.469    332.544    1.45569    0.92516
       246.000    247.034    247.061   -1.06093   -0.02729
 12    424.000    423.649    423.607    0.39279    0.04180
       249.000    248.675    248.490    0.50982    0.18484
 13    444.000    444.342    444.729   -0.72888   -0.38700
       237.000    235.674    235.603    1.39679    0.07082
 14    344.000    343.489    342.626    1.37415    0.86294
       233.000    233.566    233.889   -0.88871   -0.32252
 15    218.000    219.794    221.297   -3.29741   -1.50389
       230.000    230.837    231.851   -1.85138   -1.01422
 16     61.000     61.318     61.698   -0.69829   -0.38044
       229.000    228.864    229.171   -0.17141   -0.30691
 17     49.000     46.641     45.362    3.63817    1.27907
       434.000    433.307    434.114   -0.11445   -0.80758
 18    217.000    218.195    217.775   -0.77501    0.41995
       436.000    434.429    434.168    1.83189    0.26070
 19    347.000    346.463    348.608   -1.60841   -2.14512
       435.000    434.534    434.209    0.79116    0.32506
 20    458.000    459.112    458.552   -0.55244    0.55993
```

```
       434.000     434.148     434.243    -0.24306    -0.09530
 21    427.000     426.704     427.196    -0.19572    -0.49154
       421.000     421.907     422.133    -1.13328    -0.22628
 22    336.000     335.227     333.725     2.27532     1.50238
       422.000     422.450     422.010    -0.01049     0.43979
 23    223.000     223.840     222.829     0.17125     1.01153
       421.000     420.764     421.865    -0.86480    -1.10089
 24     76.000      75.114      77.242    -1.24183    -2.12816
       423.000     422.861     421.674     1.32646     1.18735

Uncompensated Data

Mean Squared U error 2.083018
Mean Squared V error 1.550473

Compensated Data

Mean Squared U error 1.053524
Mean Squared V error 0.304377
```

REFERENCES

Bolles, Kremers and Cain "A Simple Sensor to Gather Three-Dimensional Data", SRI Technical note 249 July 1981.

Bowman and Forrest "Transformation calibration of a camera mounted on a robot", Image and Vision Computing Vol 5, No 4 November 87. pp 261-266.

AUTHOR

Peter Kovesi

6.4 CONVERT2BYTE

NAME

convert2byte - convert any VIP image to BYTETYPE

SYNOPSIS

convert2byte [in_file] [-o out_file]

DESCRIPTION

convert2byte reads an image from stdin, or from in_file if the argument is used and converts it to BYTE (unsigned char) format. The resultant image contains values in the range 0 to 255. The conversion is a linear mapping from the input image value range to [0,255]. If the extrema of the input image are equal and less than zero, then all image values are set to zero. If the extrema of the input image are equal and greater than 255, then all image values are set to 255. If the extrema of the input image are equal and in the range [0,255] then all image values are set to this value. The output is sent to stdout unless the -o flag is used.

-o Output filename.

SEE ALSO

convert2float(1), convert2complex(1)

AUTHOR

Chris Avis

6.5 CONVERT2FLOAT

NAME

convert2float - convert any VIP image to FLOATTYPE

SYNOPSIS

convert2float [in_file] [-o out_file]

DESCRIPTION

convert2float reads an image from stdin, or from in_file if the argument is used and converts it to FLOATTYPE format. The output is sent to stdout unless the -o flag is used.

-o Output filename.

SEE ALSO

convert2byte(1), convert2complex(1)

AUTHOR

Chris Avis

6.6 CONVERT2CX

NAME

convert2cx - convert any VIP image to COMPLEXTYPE

SYNOPSIS

convert2cx [image_file] [-o outputimage_file]

DESCRIPTION

convert2cx reads an image from stdin, or from image_file if the argument is used and converts an image from byte (unsigned char), to COMPLEX format. The output is sent to stdout unless the -o flag is used.

-o Output filename.

DIAGNOSTICS

convert2cx expects a VIP image of anytype and outputs a COMPLEXTYPE image.

SEE ALSO

convert2byte(1), convert2float(1)

AUTHOR

Chris Avis.

6.7 CX2RI

NAME

cx2ri - separate the real and the imaginary components of a complex image

SYNOPSIS

cx2ri [-R real_file] [-I imag_file] [input_file]

DESCRIPTION

cx2ri reads a COMPLEXTYPE image from stdin, or from input_file if the argument is used and returns two FLOATTYPE image files: real_file and imag_file. If neither the real nor imaginary files are specified, the output is sent to stdout with firstly the content of the real_file then content of the imag_file.

-R output file name for the real component.

-I output file name for the imaginary component.

DIAGNOSTICS

cx2ri expects a COMPLEXTYPE image.

SEE ALSO

fft(1), ifft(1), mag(1), ap2cx(1), phase(1), ri2cx(1)

AUTHOR

Du Huynh

6.8 DERES

NAME

deres - deresolve image

SYNOPSIS

deres [filename] [-r resolution] [-s] [-o out_file]

DESCRIPTION

Deres reads an image of BYTETYPE from stdin, or from filename if the argument is used and deresolves the image by the resolution factor. The deresolution is accomplished by averaging over super pixels of dimension resolution by resolution. The resultant image has row and column dimensions shortened by the resolution factor. The output is sent to stdout unless the -o flag is used.

The current version of deres only works on images of BYTETYPE. Future extension of the program will be necessary to over other image types.

-r Resolution factor.

-s Select pixels from image and do not perform averaging.

-o Output filename.

DIAGNOSTICS

Deres expects an image of BYTETYPE. The resolution factor must be an integer greater than 1.

SEE ALSO

res(1), transpose(1)

AUTHORS

Chris Avis, Du Huynh

6.9 DIFFER

NAME

differ - compares two BYTETYPE images

SYNOPSIS

differ [-i BYTETYPE_in_file1] [-j BYTETYPE_in_file2] [-a -b -c] [-o BYTETYPE_out_file]

DESCRIPTION

differ compares two BYTETYPE images.

-a outputs a difference image with pixel intensity values equal to pixel intensity values in the first given image if intensity values differ in both given images. If pixel intensities in both given images are identical then the minimum pixel intensity value, (ie; 0), is assigned.

-b outputs a difference image with pixel intensity values equal to pixel intensity values in the second given image if intensity values differ in both given images. If pixel intensities in both given images are identical then the minimum pixel intensity value, (ie; 0), is assigned.

-c outputs a difference image with pixel intensity values equal to the difference in pixel intensity values between both given images.

-o output file name.

The output is sent to stdout unless the -o flag is used.

DIAGNOSTICS

differ expects two images of BYTETYPE. The images must be of identical dimensions.

AUTHOR

Du Huynh

6.10 FFT

NAME

fft - fast fourier transform of image

SYNOPSIS

fft [-s] [complex_image_file] [-o output_complex_file]

DESCRIPTION

fft reads an image of COMPLEXTYPE from stdin, or from complex_image_file if the argument is used and calculates the fast fourier transform. The row and column dimensions of the image must be an integer power of two. The output is sent to stdout unless the -o flag is used. The -s flag can be used to shift the origin of the Fourier Transform to the centre of the image.

-s set the origin shift flag on.

-o output filename.

DIAGNOSTICS

Fft expects an image of COMPLEXTYPE.

SEE ALSO

highpass(1), ifft(1), log(1), lowpass(1), mag(1), merge(1), phase(1), sever(1), unite(1)

AUTHOR

Du Huynh

6.11 GNOISE

NAME

gnoise - add gaussian noise to image

SYNOPSIS

gnoise [image_file] [-s std_dev] [-S seed] [-o out_file]

DESCRIPTION

Gnoise adds Gaussian noise of mean 0 and standard deviation sd to an image. The standard deviation sd defaults to 30. The seed value is used to start the random number generator, and defaults to 1. The output is sent to stdout unless the -o flag is used.

-s standard deviation of Gaussian Distribution.

-S seed for random number generator.

-o output file name.

DIAGNOSTICS

(1) Gnoise expects the standard deviation sd to be greater or equal to zero, values less than zero cause an error. (1) Gnoise expects an image of BYTETYPE.

AUTHOR

Chris Avis

6.12 HIGHPASS

NAME

highpass - sharpening an image in the frequency domain

SYNOPSIS

highpass [-f filter_type] [-r radius n] [-i in_file] [-o out_file]

DESCRIPTION

Highpass reads a COMPLEXTYPE image from stdin, or from in_file if the argument is used and performs either the Ideal Highpass filtering or Butterworth Highpass filtering according to the argument given in the -f flag. If the -r flag used, both the value of radius and n must be specified, although n is superfluous for the ideal filter. In_file must be a file containing a COMPLEXTYPE image created by the fft routine with the -s flag. The output is sent to stdout unless the -o flag is used. The resultant sharpened image can then be displayed by applying the ifft routine with the -s flag.

-f	filter_type = I – Ideal Highpass Filter
	filter_type = B – Butterworth Highpass Filter
-r	radius of the ring (number of pixels from the centre of the image, default = half the image size).
	n is an integer needed for the calculation in the Butterworth Filter (default = 1).
-o	Output filename.

DIAGNOSTICS

Highpass expects a COMPLEXTYPE image.

SEE ALSO

fft(1), ifft(1), lowpass(1)

AUTHOR

Du Huynh

6.13 HISTE

NAME

histe - perform histogram equalization on an image

SYNOPSIS

histe [input_image] [-o output_image]

DESCRIPTION

Histe reads in an image from stdin or from disk if input_image is specified and performs the histogram equalization process on the input image. The output image is output to stdout or written to disk if output_image is used. Histogram equalization redistributes the intensity levels by calculating statistics on the image intensity. It has the effect of bringing up low intensity information when the image is suffered from insufficient lighting.

SEE ALSO

localhisteq(1)

AUTHOR

Du Huynh

6.14 HISTOGRAM

NAME

histogram - produce a histogram data file from an image for plotting

SYNOPSIS

histogram [in_file] [-o out_file]

DESCRIPTION

Histogram reads an image from stdin, or from in_file if used and produces histogram data of the image for use by the system programs such as xgraph or xvgr. The output is sent to stdout or out_file if the -o flag is used.

-o output file name.

DIAGNOSTICS

Histogram expects various image types: BYTETYPE, RGBTYPE, and HSITYPE.

AUTHORS

Chris Avis, 1986; Du Huynh 1990, 1992.

6.15 HOMOMORPHIC

NAME

Program to perform homomorphic filtering on an image.

SYNOPSIS

homomorphic input_image LowGain HighGain CutOff order lHistogramCut uHistogramCut out_image

DESCRIPTION

Homomorphic filtering is an image enhancing process that attempts to reduce the effects of low frequency lighting variations across an image, yet at the same time accentuating high frequency intensity variations to highlight the detail. Refer to Gonzalez and Woods "Digital Image Processing" page 213.

input:

image	- input image.
LowGain	- gain factor for low frequencies (suggest 0.5).
HighGain	- gain factor for high frequencies (suggest 2.0).
CutOff	- frequency at which transition from LowGain to HighGain occurs (suggest 0.4).
order	- order of Butterworth type filter (suggest 1).
lHistogramCut	- percentage of low end of histogram to truncate (suggest 1).
uHistogramCut	- percentage of high end of histogram to truncate (suggest 5).

DIAGNOSTICS

homomorphic expects a VIP image of either BYTE, SHORT, LONG or FLOAT type.

AUTHOR

Peter Kovesi 1994.

6.16 IFFT

NAME

ifft - Inverse Fast Fourier Transform of image

SYNOPSIS

ifft [-s] [complex_image_file] [-o output_complex_file]

DESCRIPTION

Ifft reads a complex image from stdin, or from complex_image_file if the argument is used and calculates the Inverse Fast Fourier Transform. The row and column dimensions of the image must be an integer power of two. The output is sent to stdout unless the -o flag is used. If the input complex image file is created by the routine FFT with the -s flag on, it is advised that the -s flag should also be set here to undo the shifting of the origin.

-s undo the origin shift produced by the FFT routine

-o output filename.

DIAGNOSTICS

Ifft expects a COMPLEXTYPE image.

SEE ALSO

fft(1), highpass(1), log(1), lowpass(1), mag(1), merge(1), phase(1), sever(1), unite(1)

AUTHOR

Du Huynh

6.17 IMAGEINFO

NAME

imageinfo - information about a VIP image

SYNOPSIS

imageinfo [input_image]

DESCRIPTION

Imageinfo reads an image from a specified input_image containing a VIP image and returns information about that image. Returned information includes imagetype, size, dimensions, mean pixel value, high and low pixel value range, standard deviation etc.

DIAGNOSTICS

Imageinfo expects a VIP image of any type.

AUTHOR

Peter Kovesi.

6.18 IMHEAD

NAME

imhead - display header information of an image

SYNOPSIS

imhead [-c] [image_file]

DESCRIPTION

Imhead reads in a VIP image from stdin or from in_file if used and writes out the header information to stderr. If the -c flag is given and the image contains calibrated data, the camera structure stored in the image header will also be written.

-c print camera structure as well.

DIAGNOSTICS

The current version of imhead does not display the calibration matrix stored in the image header.

SEE ALSO

genimage(1)

AUTHORS

James Cooper (modified Ian Ried; Sam Mishal, 1991; Du Huynh, 1988, 1992)

6.19 KNN

NAME

knn - k-nearest-neighbour smoothing

SYNOPSIS

knn [-n nNum] [-k kNum] [in_image_file] [-o out_image_file]

DESCRIPTION

Knn reads an image from stdin, or from in_image_file if the argument is used and performs the k-nearest-neighbour smoothing process.

The size of the neighbourhood can be specified by the -n flag. The -k flag is used to specify the number of neighbours used in the smoothing calculation. The neighbourhood size is always an odd number given by (2*nNum + 1) X (2*nNum + 1) and kNum can not exceed the total number of neighbours in the neighbourhood. The smoothed FLOATTYPE image is output to stdout or written to out_image_file if the -o flag is used.

-n integer specifying the neighbourhood size.
(default neighbourhood size is 3 X 3, i.e. nNum = 1)
-k integer specifying the number of neighbours.
(default: kNum = 0.5*number of neighbours = 4)
-o output image filename.

DIAGNOSTICS

Knn expects all type of images except COMPLEXTYPE, RGBTYPE, and HSITYPE.

SEE ALSO

highpass(1), ifft(1), lowpass(1), snn(1)

AUTHOR

Du Huynh

6.20 LASER

NAME

laser - laser printer filter for VIP images

SYNOPSIS

laser [file] [-w width height] [-r nr nc] [-c n] [-n—-p position] [-e] [-lnumber label]

default: laser [stdin] -w 16 24 -c 1 -p ul

DESCRIPTION

When given a file name as an argument, laser reads the named file as an input image. The two arguments following the -r option are taken as the row and column dimensions for a raw image.

The -w arguments specify an output window on the Laser Writer. The page is specified in a row-column coordinate system, with 3180 rows and 3308 columns (A4 page). There are 300 pixels per inch. Width and height are the size of the image in centimeters.

The -c argument specifies the number of copies of the image to be printed.

The -p argument specifies the position of the image. There are 5 possible positions: ll, lr, ul, ur, and ce. Any one of these positions can be used to put the image onto the lower-left corner, lower-right corner, upper-left corner, upper-right corner or the center of the page.

The image is first scaled by the width and height factors specified after the -w flag, then translated either to the position specified by the -p argument or the default upper-left corner of the page.

If the image is to be embedded into a large document, the -n flag should be used to suppress the generation of the Postscript "showpage" command. The -n and -p flags are mutually exclusive.

If the -e flag is used, the size of the postscript files produced for synthetic images are usually smaller as the pixel values are runlength encoded. A smaller postscript file saves disk storage space but does not usually result in a shorter printing time. The LaserWriter actually takes more time to re-expand the runlength codes.

A string of characters can be superimposed on the image at a corner by using the -l flag. This is useful if there are several similar images on the same page and each image can be named individually.

Laser sends a Postscript output to stdout which can be saved to a file or piped to the line printer spooler.

Examples:

Send file scissors.im, enlarged to fit a region of size 16cm x 24cm, to the Laser Writer:

laser scissors.im — lpr -Psp

Save to file lab.ps the Postscript format of lab.im (size 7cm x 7cm) with the label "lab.im" at the bottom-right corner of the image:

laser lab.im -w 7 7 -n -l4 lab.im ¿ lab.ps

Print the VIP image stored in file room.im at the center of the page (the label "room im" is at the top-left corner of the image):

laser room.im -w 7 7 -p ce -l1 "room im" — lpr -Psp

-w Window onto A4 page.

-r Dimensions of raw image.

-c Number of copies.

-p Position of image on the page.

-n Suppress the generation of the "showpage" command.

-e Use runlength encoding.

-lnumber Put a string label at a corner of the image (see examples above).
If number = 1, label is at the top-left corner,
= 2, label is at the top-right corner,
= 3, label is at the top-left corner,
= 4, label is at the bottom-right corner.

DIAGNOSTICS

Laser expects a VIP image of BYTETYPE.

SEE ALSO

surface

AUTHORS

Chris Avis, Du Huynh

6.21 LOCALHISTEQ

NAME

Program to perform local histogram equalization.

SYNOPSIS

localhisteq [input_image] [window_size] [out_image]

DESCRIPTION

A square windowed region is indexed across the image and histogram equalization applied to the centre element of the region.

DIAGNOSTICS

localhisteq expects a VIP image of type BYTE.

SEE ALSO

histe(1)

AUTHOR

Peter Kovesi.

6.22 LOGIM

NAME

logim - apply a logarithmic transformation to a floating point image

SYNOPSIS

logim [input_file] [-o output_file]

DESCRIPTION

logim reads an image from stdin, or from input_file if the argument is used and performs the calculation: ln(1 + x) (to avoid taking the log of 0) . The output is sent to stdout unless the -o flag is used.

-o output file name to store the result.

DIAGNOSTICS

logim accepts images of all types.

SEE ALSO

expim(1), nlogim(1), sqrtim(1)

AUTHOR

Du Huynh

6.23 LOWPASS

NAME

lowpass - smoothing an image in the frequency domain

SYNOPSIS

lowpass [-f filter_type] [-r radius n] [-i in_file] [-o out_file]

DESCRIPTION

Lowpass reads a COMPLEXTYPE image from stdin, or from in_file if the argument is used and performs either the Ideal Lowpass filtering or Butterworth Lowpass filtering according to the argument given in the -f flag.

If the -r flag used, both the value of radius and n must be specified, although n is superfluous for the ideal filter.

The in_file must be a file containing a COMPLEXTYPE image created by the fft routine with the -s flag. The output is sent to stdout unless the -o flag is used.

The resultant smoothed image can then be displayed by applying the ifft routine with the -s flag.

-f	filter_type	= I – Ideal Lowpass Filter
	filter_type	= B – Butterworth Lowpass Filter
-r	radius of the ring (number of pixels from the centre of the image, default = half the image size). n is an integer needed for the calculation in the Butterworth Filter (default = 1).	
-o	output file name.	

DIAGNOSTICS

Lowpass expects a COMPLEXTYPE image.

SEE ALSO

fft(1), highpass(1), ifft(1), knn(1), snn(1)

AUTHOR

Du Huynh

6.24 MAG

NAME

mag - calculate the amplitude of a complex image

SYNOPSIS

mag [input_file] [-o output_file]

DESCRIPTION

Mag reads a COMPLEXTYPE image from stdin, or from input_file if the argument is used and returns a FLOATTYPE image that contains the magnitude (or amplitude) calculated. The output is written to output_file if the argument is given, or piped to standard output otherwise.

-o output file name for the amplitude.

DIAGNOSTICS

Mag expects a COMPLEXTYPE image.

SEE ALSO

fft(1), ifft(1), merge(1), phase(1), sever(1), unite(1)

AUTHOR

Du Huynh

6.25 NEGIM

NAME

negim - negate an image

SYNOPSIS

negim [in_file] [-o out_file]

DESCRIPTION

Negim reads an image from stdin, or from in_file if used and calculates the bitwise negation if the input is of BYTETYPE (unsigned char) or simply negate each individual pixel value if the input is of other image types. The output is sent to stdout unless the -o flag is used.

-o output file name.

DIAGNOSTICS

Negim works for all type of VIP images.

SEE ALSO

reflect(1), sumim(1), transpose(1)

AUTHORS

Chris Avis, Du Huynh

6.26 NLOGIM

NAME

nlogim - apply the natural log operation to an image

SYNOPSIS

nlogim [in_file] [-o out_file]

DESCRIPTION

nlogim reads an image from stdin, or from in_file if used and calculates a FLOATTYPE image whose elements are the natural log of the corresponding elements of the input image incremented by one. The output is sent to stdout unless the -o flag is used.

-o output file name.

DIAGNOSTICS

nlogim expects an image of BYTETYPE. Log has no restriction on the input image type but runs slower as it does not use look-up table.

SEE ALSO

expim(1), logim(1), sqrtim(1)

AUTHORS

Chris Avis, Du Huynh

6.27 PHASE

NAME

phase - calculate the phase angles of a complex image

SYNOPSIS

phase [in_file] [-o out_file]

DESCRIPTION

Phase reads a COMPLEXTYPE image from stdin, or from in_file if the argument is used and returns a FLOATTYPE image in which each pixel is the phase angles of the corresponding pixel in the input image. The output is written to out_file if the argument is given, or piped to standard output otherwise.

-o output file name for the phase angles.

DIAGNOSTICS

Phase expects a COMPLEXTYPE image and outputs a FLOATTYPE image.

SEE ALSO

fft(1), ifft(1), mag(1), merge(1), sever(1), unite(1)

AUTHOR

Du Huynh

6.28 PKSNAP

NAME

pksnap - Snap a BYTETYPE image using the PCVision+ board on a PC/AT.

SYNOPSIS

pksnap

DESCRIPTION

Pksnap is for capturing grey scale images (BYTETYPE) using the PCVision+ board on a PC/AT. As the PCVision+ board is currently installed on the 486 in the Robotics and Vision Lab, pksnap therefore needs to be run on this AT directly.

Pksnap is designed to be a user interactive program in which various options for controlling the gain and offset, selecting camera port, etc, are available.

Typing just

```
<return>
```

will print a list of available commands on the screen. Below is a short description of each of these commands:

```
ex           - exit program
snap         - freeze image
grab         - grab images continuously
gain  n      - set gain to n (0 < n < 100)
offset n     - set offset to n (0 < n < 100)
port n       - set input port to n (0 or 1)
zoom n       - set zoom factor (1 or 2)
window nr nc - set window size to nr rows and nc columns
write name   - write current window to file
stat         - get current parameter values
```

When the window command is active, the pressing of the $< up >$, $< down >$, $< left >$, or $< right >$ arrow buttons moves a rectangular window around the image. the pressing of the $< end >$ key terminates this command. When the image captured is to be saved to disk, only the portion inside the rectangular window will be considered.

The PCVision+ frame grabber has two camera ports but only the first port (port 0) has an output signal for image displaying. When port 1 is used, the display of the images being grabbed must be therefore done via a ribbon cable connected at port 0.

When the zoom factor is set to 2, the images displayed continuously on the monitor are replicated (with duplicate columns and rows). The images snapped (and saved) do not have duplicate rows and columns but the image size are only half along each dimension.

Note that only the first two characters of each command are used for recognition by the program so you can save some typing.

All Robvis software currently resides on the quokka disk which by default will mount to drive h: on the 486 via NFS at boot up time. The executable file pksnap.exe can be found in the path

```
h:\robvis\vip\bin\pcbin
```

The user is encouraged to log on to their Sun account so the image captured can be saved directly to their working directory rather than filling up the hard disk with large images. An example of the sequence of commands for logging onto your Sun account via NFS and running pksnap is given below:

```
1. net name *
The system will ask you for your login name and password.

2. net use
Once step 1 is complete, the above command returns a list
of drives currently mount to the Sun file system.
Most users should have their home directory on the drive (drive
k: on most machines) that mounts to the wambenger disk.

3. cd to the directory where you want the image to be stored

4. cd h:\robvis\vip\bin\pcbin

5. h:pksnap
   gain 0
   off 70
   sn
   wi 200 200
   wr tmp.im
```

The images that are seen by the camera can be viewed by connecting a colour monitor directly to the the camera's video output line. Depending upon how the monitor is set up, one of the lines (line A and line B) shows the images coming from the camera directly while the other shows the images via the PCVision+ frame grabber.

DIAGNOSTICS

The largest image size that can be captured is 512x512.

SEE ALSO

snap(1)

AUTHORS

Chris Avis; Du Huynh, Peter Kovesi, 1992.

6.29 PM2VIP

NAME

pm2vip - convert a PPM, PGM or PBM image to VIP format.

SYNOPSIS

pm2vip [pm image_file] [-o vip_image_file]

DESCRIPTION

pm2vip reads in a VIP image from stdin or vip_image_file if given and converts it into VIP format. The output image is written to stdout or vip_image_file if the -o flag is used.

SEE ALSO

vip2ppm(1), raster2vip(1), vip2raster(1), xdisplay(1)

AUTHOR

Carl Fisher 1992.

6.30 RASTER2VIP

NAME

raster2vip - convert a Sun raster image to the VIP format

SYNOPSIS

raster2vip [raster_image_file] [-o vip_image_file]

DESCRIPTION

raster2vip reads an image from stdin, or from raster_image_file if used and converts it to the VIP format. The output is sent to stdout unless the -o flag is used. The input image can be created by the system touchup program that runs on Sunview.

-o output file name.

DIAGNOSTICS

raster2vip expects a Sun raster image and outputs a VIP BYTETYPE image.

SEE ALSO

vip2raster(1)

AUTHOR

Du Huynh

6.31 RAW2VIP

NAME

raw2vip - convert image from raw to VIP format

SYNOPSIS

raw2vip [-t type] [-d nr nc] [image_file] [-o out_file]

DESCRIPTION

Raw2vip reads an image whose type is specified by the -t flag from stdin, or from image_file if the argument is used and converts the image from raw to standard format. This involves adding the header information to the raw image. The image type can be specified by the '-t' flag and if not given, BYTETYPE is assumed. The dimensions of the raw image are given by the -d argument. The output is sent to stdout unless the -o flag is used.

-t can be one of the following: BYTETYPE, SHORTTYPE, LONGTYPE, FLOATTYPE, DOUBLETYPE, RGBTYPE, COMPLEXTYPE, RGBTYPE, HSITYPE (default=BYTETYPE)

-d Dimensions of the image (default is 256 X 256)

-o Output filename.

SEE ALSO

vip2raw(1)

AUTHOR

Du Huynh

6.32 RI2CX

NAME

ri2cx - merge the real and imaginary images to form a complex image

SYNOPSIS

ri2cx real_file [-I imaginary_file] [-o complex_image_file]

DESCRIPTION

ri2cx reads two FLOATTYPE images from stdin, or from real_file and imaginary_file if the arguments are used and returns a COMPLEXTYPE image file.

If imaginary_file is missing, the complex image created would have the imaginary part set to zero.

If real_file is not specified, input from terminal is assumed. The output COMPLEXTYPE image is written to complex_image_file if the argument is given, or piped to standard output otherwise.

-R input filename for the real component.

-I input filename for the imaginary component.

DIAGNOSTICS

ri2cx returns a COMPLEXTYPE image.

SEE ALSO

fft(1), ifft(1), logim(1), magim(1), phase(1), cx2ri(1), ap2cx(1)

AUTHOR

Du Huynh

6.33 SCALE

NAME

scale - multiply image pixels by a scalar

SYNOPSIS

scale [filename] [-s scale] [-o out_file]

DESCRIPTION

Scale reads an image from stdin, or from filename if the argument is used and multiplies it by the scalar given after the -s argument. The output is sent to stdout unless the -o argument is used.

-s scalar value (default=1).

-o output file name.

DIAGNOSTICS

Scale expects the -s argument. Scale works for all image types.

AUTHOR

Chris Avis; Du Huynh, 1992.

6.34 SNAP

NAME

snap - Snap a BYTETYPE or RGBTYPE image using the VideoPix board on a SPARCstation.

SYNOPSIS

snap

DESCRIPTION

Snap is for capturing grey scale images (BYTETYPE) or colour images (RGBTYPE) using the VideoPix board. As the VideoPix board is currently installed on mardo, the program will need to be run on mardo by either remotely logging onto this host or logging onto the console.

Images of arbitrary size can be captured by keying in the values for image width and heigth displayed in the panel of the window. X- and Y-offset can also be changed to have a different portion of the image to be saved to disk.

DIAGNOSTICS

The colour images created by snap contain only a limited number of colours and grey scale images have less than 128 grey levels. The quality of the images produced is also poorer than that produced by pcsnap that works on PC/ATs.

SEE ALSO

pksnap(1)

AUTHOR

Du Huynh

6.35 SNN

NAME

snn - symmetric-nearest-neighbour smoothing an image

SYNOPSIS

snn [-n num] [input_image] [-o output_image]

DESCRIPTION

Snn reads an image from stdin, or from input_image if the argument is used and performs the symmetric-nearest-neighbour smoothing process.

The size of the neighbourhood can be specified by the -n flag. The neighbourhood size is always an odd number given by (2*nNum + 1) X (2*nNum + 1). One of each pair of neighbouring pixels symmetrically opposite to the central pixel is chosen in the averaging process with the chosen criteria being the nearness of the pixel value to the central pixel value. So exactly half of the neighbouring pixels will be chosen and the pixel value is replaced by the calculated average. The output smoothed FLOATTYPE image is directed to stdout or written to output_image if the -o flag is used.

-n integer specifying the neighbourhood size.
(default neighbourhood size is 3 X 3, i.e. nNum = 1)
-o output FLOATTYPE image file name.

DIAGNOSTICS

Snn expects a BYTETYPE image.

SEE ALSO

knn(1), lowpass(1)

AUTHOR

Du Huynh

6.36 SPLASH

NAME

splash - image displaying program for X Windows.

SYNOPSIS

splash [imagename]

DESCRIPTION

Splash displays VIP format images to 128 grey levels. BYTE, SHORT, LONG, FLOAT, DOUBLE and COMPLEX image types can be displayed. In the case of complex images one can select to see either the magnitude or the phase of the image.

A number of operations are available under the tools control frame.

magnification:

The image can be magnified by clicking on the desired magnification button. The section of the image that is displayed in the window can be adjusted by dragging the red display window box in the preview window to the desired position.

xyz to uv conversion:

Entering the x, y, and z values of a 3D point in this field will result in the corresponding image position being marked by a small red square.

uv to xyz conversion:

Enter a plane.list file name and a plane number to use. The image position of the cursor will be converted to a 3D position using the constraint plane that you specified. See manual entry for vipcalib for more information.

Gamma adjustment:

This slider adjusts a look up table used to display the image. The table is generated using the function Set_Look_Up_Table (in vip1D.c). The grey values are normalised from 0-255 to 0-1, and then raised to a power 'gamma'. Gamma is obtained by dividing the slider value by 100. A gamma value of 1 (100 on the slider) produces a linear table resulting in the normal display. A value less than 1 will expand the dark end of the scale and compress the bright end (useful for suppressing wool texture).

Digitising slices:

Digitise start and end points of the slice using the left mouse button. Two plots of the intensity values along the slice are then displayed. One is a plot of raw intensity values and the other is a plot of values mapped through the look up table generated via the 'gamma' slider. This is useful for determining values of gamma that are likely to make the detection of various features easier.

DIAGNOSTICS

Splash expects images in the VIP format (new or old). These may be supplied at the Unix command line or within the Splash application.

BUGS

Processing of parameters supplied to the xyz to uv and uv to xyz commands is a bit fragile.

AUTHOR

Peter Kovesi

6.37 SPLM

NAME

splm - vip image displaying program, using Motif. (Currently on DEC Alpha only).

SYNOPSIS

splm [vip BYTETYPE image] [vip FLOATTYPE image] [vip COMPLEXTYPE image]

DESCRIPTION

Splm displays images to 128 grey levels. The image can be magnified as desired.

A number of operations are available under the tools control frame.

Magnification factor:

Using the toggles provided, the image being displayed may be magnified by factors 1, 2, 3, 4, 5 and 6.

Using the toggle provided, the previously displayed image may be recalled.

Threshold Value:

A value between 0 and 255, by which a threshold level is set, is selected by via a slider. Any pixel with a grey value below this threshold value will be given a grey value of 0, else the pixel will be given a grey value of 255. In this way a monochrome image is created. There is a toggle next to the slider by which the thresholding feature can be activated or deactivated. A default value of 100 is initially assumed.

Gamma adjustment:

This slider adjusts a look up table used to display the image. The table is generated using the function Set_Look_Up_Table (in vip1D.c). The grey values are normalised from 0-255 to 0-1, and then raised to a power 'gamma'. Gamma is obtained by dividing the slider value by 100. A gamma value of 1 (100 on the slider) produces a linear table resulting in the normal display. A value less than 1 will expand the dark end of the scale and compress the bright end (useful for suppressing wool texture). There is a toggle next to the slider by which the gamma value may be reset. The default value of 1 is initially assumed.

Digitising slices:

Digitise start and end points of the slice using the left mouse button. Two plots of the intensity values along the slice are then displayed. One is a plot of raw intensity values and the other is a plot of values mapped through the look up table generated via the 'gamma' slider. This is useful for determining values of gamma that are likely to make the detection of various features easier.

AUTHORS

Peter Kovesi, Carl Fisher

6.38 SPY

NAME

spy - view image data values

SYNOPSIS

spy [file] [-p] [-i initial_row] [-f final_row]

DESCRIPTION

Spy reads an image from stdin, or from file if the argument is used and writes the data in ascii form to stdout. By default the entire row range of an image is displayed. The displayed row range can be modified by the use of the -i and -f arguments. The -p argument is used to view the data in double precision.

-i initial row to be displayed.

-f final row to be displayed.

-p double precision output.

DIAGNOSTICS

Spy works for all type of V.I.P. images. The -i and -f arguments must be integer values in the image row domain.

SEE ALSO

hist(1), stat(1)

AUTHORS

Chris Avis; Du Huynh 1992.

6.39 SQRIM

NAME

sqrim - square pixel values in an image

SYNOPSIS

sqrim [input_file] [-o output_file]

DESCRIPTION

sqrim reads an image from stdin, or from input_file if used and calculates a FLOATTYPE or DOUBLETYPE image whose elements are the square of the corresponding elements of the input image. The output is sent to stdout unless the -o flag is used.

-o output file name.

DIAGNOSTICS

sqrim expects an image of FLOATTYPE or DOUBLETYPE. The output image type is kept identical to that of the input image.

SEE ALSO

sqrtim(1), sumim(1)

AUTHORS

Chris Avis; Du Huynh, 1992.

6.40 SQRTIM

NAME

sqrtim - take the square root of pixel values in an image

SYNOPSIS

sqrtim [input_file] [-o output_file]

DESCRIPTION

sqrtim reads a FLOATTYPE or DOUBLETYPE image from stdin, or from input_file if used and calculates an image whose elements are the square root of the corresponding elements of the input image. The output is sent to stdout unless the -o flag is used.

-o output file name.

DIAGNOSTICS

sqrtim expects an image of FLOATTYPE or DOUBLETYPE. The output image has the same image type as the input image.

SEE ALSO

sumim(1), sqrim(1)

AUTHOR

Chris Avis

6.41 SUMIM

NAME

sumim - sum two matrices

SYNOPSIS

sumim [image_1] [image_2] [-o sum_image]

DESCRIPTION

Sumim reads two images from stdin, or from image_1 and image_2 if the arguments are used. If image_2 is not given on the command line, it is assumed to be from stdin. If both image_1 and image_2 are not given, they are assumed to be from stdin also. Sumim pixelwisely adds these two images are writes the output to stdout unless the -o flag is used.

-o output file name.

DIAGNOSTICS

Sumim works for all type of VIP images.

SEE ALSO

negim(1), sqrim(1), sqrtim(1)

AUTHOR

Chris Avis; Du Huynh, 1992.

6.42 SURFACE

NAME

surface - display a VIP image as a 2D surface

SYNOPSIS

surface

DESCRIPTION

Surface is a XView program that allows the user to display a VIP image as a two-dimensional surface by treating the pixel values as the height of the surface patches.

Surface has the following buttons in its panel:

1. Quit. To exit the program.

2. Close. To close the window to an icon.

3. Load. To load an image whose file name is given in the input text field File name.

4. Reset Transformation. Reset the transformation to that given by the pitch and yaw angles.

5. Print. To bring up a sub-menu that enable the sending of the current pose of the 2D surface to disk or to the printer.

6. Help. Print out a help message about the program.

7. File name. a text field for the user to key in the image file name.

8. Pitch. The pitch angle can be controlled by keying in an integer value in the range [-180,180] or the dragging of the slider bar.

9. Yaw. The yaw angle can be controlled by keying in an integer value in the range [-180,180] or the dragging of the slider bar.

The surface printing sub-menu contains the following fields:

1. Postscript format. The user can select the postscript file to be either directly printable or encapsulated.

2. Print to device. The postscript file to be generated by surface can be saved to disk if a file name is given or sent to the printer directly.

3. File name. Only used if Disk is used in field 2 above.

4. Printer name. Only used if Printer is used in field 2 above.

5. Picture height (cm). The height of the picture in cm to appear on the page. Surface will scale the picture appropriately to the height specified while reserving the picture's aspect ratio.

6. Cancel. To cancel the print sub-menu.

7. Print. This button should only be selected after all the above fields have been carefully chosen.

If the postscript is to be included in a document, both encapsulated (field 1) and Disk (field 2) should be used and File name (field 3) should be specified.

If the postscript is to send to the printer then Printable (field 1), Printer (field 2) and Printer name should be used.

All the buttons in the panel can be activated by the pressing of the left button of the mouse.

DIAGNOSTICS

Every update of the pitch or yaw angle updates the transformation automatically in a cumulative manner. It is usually difficult therefore to get back to a pose the 2D surface was at previously after a number of changes on the pitch and yaw values. The reset transformation option allows the user to reinitialize the transformation to an identity matrix and then update it using the selected pitch and yaw.

Surface accepts only images of BYTETYPE, SHORTTYPE, LONGTYPE, FLOATYPE, and DOUBLETYPE.

SEE ALSO

splash, splm, xdisplay

AUTHOR

Du Huynh

6.43 THRESH

NAME

thresh - threshold image

SYNOPSIS

thresh [input_file] [-t|-f|-c threshold] [-o output_file]

DESCRIPTION

Thresh reads an image from stdin, or from input_file if the argument is used and thresholds the image by the threshold value.

If the -t flag is used, all pixel values less than the threshold value are set to "0", and all values greater than or equal to the threshold value are set to "255".

If the -f flag (floor thresholding) is used, the thresholding process will zero out those pixels below the threshold value and leave other pixels unchanged.

If the -c flag (ceiling thresholding) is used, pixel values above the given threshold value will be turned to zeros instead.

If none of these flags is used, the -t flag is assumed and the default threshold value is "128".

If the given threshold value is a floating point number less than 1, it is assumed that threshold represents the percentage of pixels in the image that needs to be zeroed out. A corresponding threshold calculated from the image statistics will then be applied.

The output is sent to stdout unless the -o flag is used.

-t threshold value.

-o output file name.

DIAGNOSTICS

Thresh expects an image of BYTETYPE.

Threshold must be an integer in the range 0 to 255 or a floating number between 0 and 1.

SEE ALSO

histogram

AUTHORS

Chris Avis, Du Huynh

6.44 TRANSPOSE

NAME

transpose - transpose an image

SYNOPSIS

transpose [input_file] [-o output_file]

DESCRIPTION

Transpose reads an image from stdin, or from input_file if the argument is used and transposes it (similar to matrix transpose in Linear Algebra). The output is sent to stdout unless the -o flag is used.

-o output file name.

DIAGNOSTICS

The current version of transpose works only on images of BYTETYPE.

SEE ALSO

reflect(1)

AUTHOR

Du Huynh

6.45 VIP2PPM

NAME

vip2ppm - convert a VIP image to PPM format

SYNOPSIS

vip2ppm [vip_image_file] [-o ppm_image_file]

DESCRIPTION

Vip2ppm reads in a VIP image from stdin or vip_image_file if given and convert it into PPM format. The output image is written to stdout or ppm_image_file if the -o flag is used.

Vip2ppm only supports BYTETYPE and RGBTYPE images. Images of other type will need an extra conversion routine before they can be passed to this program.

To save disk space, vip2ppm always creates PPM files in binary form.

SEE ALSO

raster2vip(1), vip2raster(1), xdisplay(1)

AUTHOR

Du Huynh

6.46 VIP2RASTER

NAME

vip2raster - convert a VIP image to the Sun raster format

SYNOPSIS

vip2raster [vip_image_file] [-o raster_image_file]

DESCRIPTION

vip2raster reads an image from stdin, or from vip_image_file if used and converts it to a Sun raster image. The output is sent to stdout unless the -o flag is used. The output image format is consistent with the format used in the system touchup program.

-o output file name.

DIAGNOSTICS

vip2raster expects an image of BYTETYPE or RGBTYPE.

SEE ALSO

raster2vip(1)

AUTHOR

Du Huynh

6.47 VIP2RAW

NAME

vip2raw - strip off image header from VIP image

SYNOPSIS

vip2raw [input_file] [-o output_file]

DESCRIPTION

Vip2raw reads an image from stdin, or from input_file if the argument is used and converts the image from standard VIP format to raw format. This involves removing the header information from the standard image. The output is sent to stdout unless the -o flag is used.

-o output file name.

DIAGNOSTICS

Vip2raw works for all type of images.

SEE ALSO

raw2vip(1)

AUTHOR

Du Huynh

6.48 XDISPLAY

NAME

xdisplay - display a VIP image in X-Windows.

SYNOPSIS

xdisplay image_file1 image_file2...

DESCRIPTION

Xdisplay reads in a list of images from stdin or from image_file1, image_file2, etc. if any filenames are given and displays it/them on the X-Windows environment. If a FLOATTYPE image is to be shown, it must be first converted to a BYTETYPE image by use of fc. Xdisplay forks off one process to display each image and returns immediately. Each image is displayed in its own window which remains active until the window is destroyed.

The current version of xdisplay converts the input VIP image into PPM (portable pixmap) format, xv is then called to actually create the window and display the image. This allows xdisplay to work on X-Window terminals which have monochrome screen using the dithering technique implemented in xv.

OPTIONS

Right button: clicking the right button when the mouse is in the canvas turns ON/OFF a sub-menu which contains various options such as enlarging/diminishing the image in both dimensions by 10rotating the image anti-clockwisely or clockwisely, enlarging a selected portion of the image, bringing up another sub-menu for gamma adjustment, saving the image in various available format and displaying new images, etc.

Left button: clicking the left button while the mouse is in the canvas returns the current pixel position and pixel value.

Middle button: dragging the middle button while the mouse is in the canvas enables the selection of a portion of the image for enlargement, etc.

DIAGNOSTICS

Xdisplay expects an image of BYTETYPE or RGBTYPE.

Xdisplay works only on X-Windows.

SEE ALSO

splash, splm, surface

AUTHOR

Du Huynh

6.49 XYSCALE

NAME

Program to rescale images with arbitrary x and y scalings.

SYNOPSIS

xyscale [input_image] [x_scaling_factor] [y_scaling_factor] [out_image]

DESCRIPTION

xyscale reads an image from the specified input file containing a VIP image and scales it according to the x and y scaling factors, (which may be real number). A suitably scaled image is output.

DIAGNOSTICS

xyscale expects a VIP image of types BYTE, SHORT, LONG and FLOAT only.

AUTHOR

Peter Kovesi.

Chapter 7

VIP Library Function Reference

7.1 VIP Input/Output and Memory Management Functions

Allocate_Header, Allocate_Image, Free_Image, Read_Header, Write_Header, Read_Image, Write_Image, Copy_Header, Print_Header, Write_VFFImage, Read_PCImage, VIP2PPM_Image

#include "vip.h"

IMAGE *Allocate_Header(int umin, int vmin, int rows, int cols, int type);

Allocate_Header allocates memory for an image header. The content of the image itself is not allocated. Returns NULL if unsuccessful.

IMAGE *Allocate_Image(int umin, int vmin, int rows, int cols, int type);

Allocate_Image allocates memory for an image of the desired size and type and returns the address. Returns NULL if unsuccessful.

int Free_Image(IMAGE *image);

Free_Image frees memory allocated for an image.

IMAGE *Copy_Image(IMAGE *image);

Copy_Image creates a duplicate copy of an existing image.

int Read_Header(IMAGE *image, FILE *fp);

Read_Header reads in only the header part of an image to the given file pointer. Read_Header does byte swapping if the macro SUN is not defined (for PC/AT programs).

int Write_Header(IMAGE *image, FILE *fp);

Write_Header writes an image header to the given file pointer. Similar to Read Header. Write_Header also does byte swapping if the macro SUN is not defined.

IMAGE *Read_Image(char *filename);

Read_Image reads an image from a file, allocates memory for it and returns the address. Returns NULL if unsuccessful.

int Write_Image(IMAGE *image, char *filename);

Write_Image writes an image to a file. Returns OK (1) if successful, ERROR (0) if not.

int Copy_Header(IMAGE *image1, IMAGE *image2);

Copy_Header copies the header information from image1 to image2 (not including the pointer to the image data itself).

void Print_Header(IMAGE *image)

Print_Header prints out the header information of an image.

int Write_VFFImage(IMAGE *image, char *filename);

Write_VFFImage converts a VIP image into vff format and writes it to a file for display under SUNVISION.

IMAGE *Read_PCImage(char *filename)

Read_PCIMage reads a binary file written by the PC and converts it into VIP format. Normally you do not have to use this function as it is automatically invoked by Read Image when it finds the image it is trying to read is not already in VIP format.

int VIP2PPM_Image(IMAGE *image, char *filename);

VIP2PPM_Image saves a VIP image to a file in PPM format and returns OK (1) if the process is successful or returns ERROR (0) otherwise.

7.2 Image Type and Format Conversion Functions

Convert2Byte_Image, Convert2Short_Image, Convert2Long_Image, Convert2Float_Image, Convert2Double_Image, Convert2Complex_Image, Complex2PhaseFloat_Image, Complex2MagFloat_Image, Construct_Complex_Image, Extract_Real, Extract_Imaginary

#include "vip.h"

IMAGE *Convert2Byte_Image(IMAGE *image);

Convert2Byte Image converts an image of SHORT, LONG, FLOAT or DOUBLE TYPE to a BYTETYPE image and returns the address of the new image. Image values are rescaled to the range 0-255 and then cast to BYTE type. Returns NULL if unsuccessful.

IMAGE *Convert2Short_Image(IMAGE image);

Convert2Short Image converts an image of BYTE, LONG, FLOAT or DOUBLE TYPE to a SHORTTYPE image and returns the address of the new image. No range checks for overflow are made. Returns NULL if unsuccessful.

IMAGE *Convert2Long_Image(IMAGE *image);

Convert2Long Image converts an image of BYTE, SHORT, FLOAT or DOUBLE TYPE to a LONGTYPE image and returns the address of the new image. No range checks for overflow are made. Returns NULL if unsuccessful.

IMAGE *Convert2Float_Image(IMAGE *image);

Convert2Float Image converts an image of BYTE, SHORT, LONG or DOUBLE TYPE to a FLOATTYPE image and returns the address of the new image. No range checks for overflow are made. Returns NULL if unsuccessful.

IMAGE *Convert2Double_Image(IMAGE *image);

Convert2Double Image converts an image of BYTE, SHORT, LONG, or FLOAT TYPE to a DOUBLETYPE image and returns the address of the new image. No range checks for overflow are made. Returns NULL if unsuccessful.

IMAGE *Convert2Complex_Image(IMAGE *image);

Convert2Complex converts an image of BYTE, SHORT, LONG, FLOAT or DOUBLE TYPE to a COMPLEXTYPE image and returns the address of the new image. No range checks for overflow are made. Returns NULL if unsuccessful.

IMAGE *Complex2PhaseFloat_Image(IMAGE *image);

Complex2PhaseFloat converts a COMPLEXTYPE image to a FLOAT type Phase image and returns the address of the new image. No range checks for overflow are made. Returns NULL if unsuccessful.

IMAGE *Complex2MagFloat_Image(IMAGE *image);

Complex2MagFloat Image converts a COMPLEXTYPE image to a FLOAT type Magnitude image and returns the address of the new image. No range checks for overflow are made. Returns NULL if unsuccessful.

IMAGE *Construct_Complex_Image(IMAGE *Re, IMAGE *Im);

Function to construct a COMPLEX TYPE image from two FLOAT TYPE images, one being the real and the other the imaginary component of the new image.

IMAGE *Extract_Real(IMAGE *image);

Function to extract the real part of a complex image. This is returned as a FLOAT image.

IMAGE *Extract_Imaginary(IMAGE *image);

Function to extract the imaginary part of a complex image. This is returned as a FLOAT image.

int VIP2PPM_Image(IMAGE *image, char *filename);

See VIP Input Output Functions.

IMAGE *Read_PCImage(char *filename)

See VIP Input Output Functions.

7.3 VIP Image Enhancement and Spatial Transformation Functions

Bilinear_Interpolate, XYScale_Image, Rotate_Image, Truncate_Image_Histogram, Thresh_Image, Floor_Thresh_Image, Ceiling_Thresh_Image, Gaussian_1D, Gaussian_2D, Laplacian_Gaussian_1D, Laplacian_Gaussian_2D, Apply_Local_Energy, Convolve_Image, FConvolve_Image, Point_Convolve, DeRes_Image, Res_Image, Knn_Image, Snn_Image, Clip_Image, Image_Row_Reflect, Image_Column_Reflect, Set_Look_Up_Table, Erode_Image, Dilate_Image, Allocate_Float_Mask, Free_Float_Mask, Float_Mask2Image, Scale_Float_Mask, Rotate_Float_Mask, Read_Int_Mask, Allocate_Int_Mask, Write_Int_Mask, Read_Energy_Filter, Image_Point, Image_Line, Distance_Transform

#include "vip.h"

double *Bilinear_Interpolate(IMAGE *inimage, double x, double y);

inimage - image to interpolate.
x,y - real valued location of desired interpolated grey level.

Function to perform bilinear interpolation of grey values in an image. Function returns the interpolated grey value.

IMAGE *XYScale_Image(IMAGE *inimage, float xscale, float yscale);

Function to scale an image in x and y directions.
Returns scaled version of image.

IMAGE *Rotate_Image(IMAGE *inimage, double angle);

Function to rotate an image. Bilinear interpolation is used to calculate the grey values.

inimage - input image.
angle - rotation angle in radians.

Returns rotated version of image.

IMAGE *Truncate_Image_Histogram(IMAGE *image, int lowerpercent, int higherpercent);

Function to modify an image so that specified percentages of the extremes of its histograms are truncated. Values in the image are clamped to lie within this range resulting in some saturation in the image. The aim is to remove outlying pixel values from an image so that its displayed appearance is improved.

IMAGE *Thresh_Image(IMAGE *image, int threshold);

image - input image.
threshold - threshold value.
Returns an image thresholded at a given value.

IMAGE *Floor_Thresh_Image(IMAGE *image, int threshold);

Zeroes out those pixels whose values are lower than the given threshold value but leaves other pixel values unchanged. This function performs the opposite operation as that in function Ceiling_Thres Image. The function returns the thresholded image. Returns NULL if unsuccessful.

IMAGE *Ceiling_Thresh_Image(IMAGE *image, int threshold);

Zeroes out those pixels whose values are higher than the given threshold value but leaves other pixel values unchanged. This function performs the opposite operation as that in function Floor Thres Image. The function returns the thresholded image. Returns NULL if unsuccessful.

IMAGE *Gaussian_1D(int omega);

Produces a one-dimensional Gauss FLOATTYPE image. omega denotes the spread of the Gauss function. Returns NULL if unsuccessful.

IMAGE *Gaussian_2D(int omega);

Produces a two-dimensional Gauss FLOATTYPE image. omega denotes the spread of the Gauss function. Returns NULL if unsuccessful.

IMAGE *Laplacian_Gaussian_1D(int omega);

Produces a one-dimensional Laplacian of a Gaussian FLOATTYPE image. omega denotes the spread of the Gauss function. Returns NULL if unsuccessful.

IMAGE *Laplacian_Gaussian_2D(int omega);

Produces a two-dimensional Laplacian of a Gaussian FLOATTYPE image. omega denotes the spread of the Gauss function. Returns NULL if unsuccessful.

IMAGE *Apply_Local_Energy(IMAGE *image, float *Hmask, float *Fmask, int mask size, float threshold);

Apply the Local Energy feature detector to an image. The process involves convolving the given image with an odd and an even masks of a given size. The argument 'threshold' is for the non-maxima suppression process. The function returns the convolution output or NULL if unsuccessful.

IMAGE *Convolve_Image(IMAGE *image, INT_MASK *mask);

This function performs 2D convolution with a mask of integers. To achieve precision the integer mask values can be scaled up by a rescale factor. The final convolution result is then divided by this rescale factor to correct the result.

For example: to perform a 3x3 average, the mask values would be 1 and the rescaling parameter would be 9.

Convolve Image takes BYTETYPE, SHORTTYPE and LONGTYPE images only. Returns NULL if unsuccessful.

IMAGE *FConvolve_Image(IMAGE *image, FLOAT_MASK *mask);

Floating Point Convolution Function. This function performs a 2D convolution with a mask of floats. A pointer to a FLOAT image is returned. Valid for BYTE, SHORT, LONG and FLOAT images.

float *Point_Convolve(IMAGE *image, int uc, int vc, FLOAT_MASK *mask);

Function returns the result of convolving a FLOAT MASK with an image at a specified point.

IMAGE *DeRes_Image(IMAGE *image, int factor);

Function to reduce the resolution of an image by a value of 'factor'. Processes only BYTEYTPE images. Return NULL if unsuccessful.

IMAGE *Res_Image(IMAGE *image, int factor, int average flag);

Function to increase the resolution of an image by a factor given in the second parameter. If the factor is 2 and the last parameter is non-zero, a Gaussian process (7x7 mask) is applied to smooth the output image. The function processes only BYTEYTPE images and returns NULL if unsuccessful.

IMAGE *Knn_Image(IMAGE *image, int n, int k);

Returns a K Nearest Neighbour Smoothing image and NULL if unsuccessful. The size of the neighbourhood used in the smoothing process is (2*n+1) x (2*n+1). K is the number of neighbours to be selected in the smoothing process. Knn Image handles all type of images except COMPLEXTYPE, RGBTYPE, and HSITYPE. The output image is of FLOATTYPE.

IMAGE *Snn_Image(IMAGE *image, int neighbour int size);

Returns a Symmetric Nearest Neighbour Smoothing of an image and NULL if unsuccessful. The size of the neighbourhood used in the smoothing process is nbhsiz x nbhsize. Snn Image handles all type of images except COMPLEXTYPE, RGBTYPE, and HSITYPE. The output image is of FLOATTYPE.

IMAGE *Clip_Image(IMAGE *image, int origin r, int origin c, int num rows, int num cols);

Returns a sub-image from a larger image. All the arguments are assumed to be within the image boundary. The coordinates of the top-left corner of the image are specified by r and c, num rows and num cols are the height and width of the output image respectively.

IMAGE *Image_Row_Reflect(IMAGE *image);

Returns an image that is a horizontal mirror reflection of a given image or NULL if unsuccessful. The function handles all image types.

IMAGE *Image_Column_Reflect(IMAGE *image);

Returns an image that is a vertical mirror reflection of a given image or NULL if unsuccessful. The function handles all image types.

int Set_Look_Up_Table(int LUT[256], double gamma);

Function to set up a lookup table for modifying intensity values using gamma correction.

IMAGE *Erode_Image(IMAGE *inimage, INT_MASK *mask);

Function to perform morphological erosion using a mask specified in an INT_MASK structure. An example of a 5x5 morphological mask that approximates a circular disc would be:

```
* morphological mask

0 1 1 1 0
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
0 1 1 1 0
```

IMAGE *Dilate_Image(IMAGE *inimage, INT_MASK *mask);

Function to perform morphological dilation using a mask specified in an INT_MASK structure.

IMAGE *Distance_Transform(IMAGE *inimage);

Function computes a distance transform for a binary image. It is assumed that objects pixels in the image have a value of 0.

Function returns a SHORTTYPE image with distances away from the objects calculated using the distance relationship shown below.

```
-------------
| 7 | 5 | 7 |
-------------
| 5 | x | 5 |
-------------
| 7 | 5 | 7 |
-------------
```

7.4 VIP Mask Functions

INT_MASK *Read_Int_Mask(char *filename);

Function to read a mask file into memory.

The expected file format is:

Blank lines can used anywhere. Comment lines start with a '*'.

Data format is simply 2D array of integer mask values (For morphological masks a zero value indicates a non active element. The optional 'rescale' parameter is used when the integer mask values have been scaled up to provide extra precision. After the convolution has been performed the result can be scaled down by this parameter to normalise the result. If the rescale value is omitted it is assumed to be 1. The mask need not be square but should be rectangular, eg.

```
* 3 x 3 Sobel mask for vertical edges

rescale 1
-1 0 1
-2 0 2
-1 0 1
```

BUG WARNING: Do not leave any trailing blanks on any lines that specify the mask. Read_Int_Mask does not handle these correctly.

Convolutions and morphological operations are generally specified using an INT_MASK structure. This structure is simply a 2D array of integers which specifies the mask. This mask structure can be used for 1D or 2D convolutions. (1D convolution indicated by having one row). The mask data is specified using integers for fast calculation. It is intended that in order to obtain good precision the ideal, real valued, mask values would be scaled up by some factor so that their integer representation is reasonable. The convolution would then be performed with these scaled up masks, and the final result divided by "rescale" to normalise the result.

When this structure is used to represent morphological masks a zero value designates an inactive element.

The structure is listed below

```
typedef struct {
    int      idcode;
    char     version[8];
    int      rows;
    int      cols;
    int      rescale;
    l array *m;                  /* mask data */
}        INT_MASK;
```

Elements of a mask are accessed simply with a statement of the form

```
value = mask->m[row][col];
```

INT_MASK *Allocate_Int_Mask(int rows, int cols);

Function to allocate space for an INT_MASK structure.

void Free_Int_Mask(INT_MASK *mask);

Function to free memory allocated for an INT_MASK structure.

int Write_Int_Mask(INT_MASK *mask, char *filename, char * comment);

Function writes a INT_MASK structure to a file.

void *Allocate_Float_Mask(int rows, int cols);

Function to allocate space for a FLOAT_MASK structure. The FLOAT_MASK structure is listed below. The symmetry fields are intended to allow one to optimize convolution operations, depending on the structure of the mask one can reduce the number of multiplication operations by up to a factor of 4.

```
#define FLOAT_MASK_ID 1
#define NOT_SYMMETRIC 0
#define SYMMETRIC 1
#define ANTI_SYMMETRIC -1

typedef struct struct_vip_FLOAT_MASK {
    int     idcode;
    char    version[8];
    int     rows;
    int     cols;
    int     xsymmetry;
```

```
    int      ysymmetry;
    f_array *m;                  /* mask data */
}       FLOAT_MASK;
```

void *Free_Float_Mask(FLOAT_MASK *mask);

Function to free memory allocated for a FLOAT_MASK structure.

IMAGE *Float_Mask2Image(FLOAT_MASK *mask, int rows, int cols, int shift);

Copy a FLOAT_MASK to a FLOAT image of size rows x cols so that one can use its FFT for convolution with an image.

If shift is 0 the mask is placed around the 'corners' of the image in the default FFT data arrangement.

If shift is 1 the mask is centred in the image.

FLOAT_MASK *Scale_Float_Mask(FLOAT_MASK *mask, float scale);

mask - reference mask.
scale - mask scaling factor.

Function to scale values in a FLOAT_MASK.

FLOAT_MASK *Rotate_Float_Mask(FLOAT_MASK *inmask, double angle);

inmask - reference mask.
angle - rotation angle in radians.

Function returns a rotated version of a reference mask.

int Read_Energy_Filter(char *filename, float **Hmask, float **Fmask, int *mask size);

Read_Energy_Filter reads from disk an odd and an even filter for the Local Energey feature detector. The function returns 1 for successful read, 0 otherwise.

7.5 VIP Graphics Functions

void *Image_Point(IMAGE *image, int u, int v, int size, int grey);

Function to draw a point in an image.

void *Image_Line(IMAGE *image, int us, int vs, int ue, int ve, int grey);

Function to draw a line in an image.

7.6 VIP Image Arithmetic Functions

Set_Image, Average_Image, Diff_Image, Subtract_Image, Add_Image, Exp_Image, Log_Image, Logarithmic_Transform_Image, Squared_Image_Difference, Scale_Image, Offset_Image, Sqrt_Image, Sqr_Image, Divide_Image, Multiply_Image, Accumulate_Add_Image

#include "vip.h"

int Set_Image(IMAGE *image, double value);

Set Image sets an entire image to the specified value, performing the appropriate casting depending on the image type. For COMPLEX, RGB and HSI types, all fields are set to the value specified. Returns OK (1) if successful, ERROR (0) if not.

IMAGE *Average_Image(IMAGE *image1, IMAGE *image2);

Function takes the average of two images, returning the address of the new image.

IMAGE *Diff_Image(IMAGE *image1, IMAGE *mage2, int gain);

Function to take the absolute difference of two images. Difference is amplified by gain. Supports BYTE, SHORT, LONG and FLOAT images. In the case of BYTE images the result is clamped to 0-255.

IMAGE *Subtract_Image(IMAGE *image1, IMAGE *image2, int gain, int offset);

Function to subtract two images. Image2 is subtracted from image1, the result is multiplied by 'gain' and shifted by 'offset'. Supports BYTE, SHORT, LONG and FLOAT images. In the case of BYTE images the result is clamped to 0-255.

IMAGE *Add_Image(IMAGE *image1, IMAGE *image2);

Function to add two images. Valid for BYTE, SHORT, LONG and FLOAT images. In the case of BYTE images the result is clamped to 0-255.

IMAGE *Exp_Image(IMAGE *image);

Function to take the natural exponent of an image. Input images are converted to FLOATTYPE, the natural exponent taken, and a FLOATTYPE image is returned.

IMAGE *Log_Image(IMAGE *image);

Function to take the natural log of an image. Input images are converted to FLOATTYPE, the natural log taken, and a FLOATTYPE image is returned. Note that 1 is added to each pixel value to avoid taking the log of 0.

IMAGE *Logarithmic_Transform_Image(IMAGE *image);

Returns the logarithmic transform of an image. Each pixel is incremented by one prior the application of the logarithmic function. Logarithmic Transform Image works for all image types. Returns NULL if unsuccessful.

IMAGE *Squared_Image_Difference(IMAGE *image1, IMAGE *image2);

Function returns a FLOAT image that is the squared difference between two input images. Valid for BYTE, SHORT, LONG, FLOAT and DOUBLE images.

IMAGE *Scale_Image(IMAGE *image, float factor);

Function to pixel-wise scale an image by a given factor. Works for all image types except RGBTYPE and HSITYPE. Returns NULL if unsuccessful.

IMAGE *Offset_Image(IMAGE *image, float factor);

Function to add an offset to each pixel in an image by a given factor. Works for all image types except COMPLEXTYPE, RGBTYPE and HSITYPE. Returns NULL if unsuccessful.

IMAGE *Sqrt_Image(IMAGE *image);

Returns an image whose pixel values are square roots of the corresponding pixel in the input image. Valid for BYTE, SHORT, LONG, FLOAT and DOUBLETYPE images. Returns NULL if unsuccessful.

IMAGE *Sqr_Image(IMAGE *image);

Returns an image whose pixel values are squares of the corresponding pixel in the input image. Valid for BYTE, SHORT, LONG, FLOAT and DOUBLETYPE images. In the case of BYTE images the result is clamped to 0-255. Returns NULL if unsuccessful.

IMAGE *Divide_Image(IMAGE *image1, IMAGE *image2);

Function to pixelwise divide two images. There are no checks for numerical overflow. Valid for BYTE, SHORT, LONG, FLOAT, DOUBLE and COMPLEX images.

IMAGE *Multiply_Image(IMAGE *image1, IMAGE *image2);

Function to pixelwise multiply two images together. There are no checks for numerical overflow. Valid for BYTE, SHORT, LONG, FLOAT, DOUBLE and COMPLEX images.

void Accumulate_Add_Image(IMAGE *Ac, IMAGE *image);

Function to perform an accumulation addition of an image to another. Elements in image Ac are incremented by the corresponding values in image. There are no checks for numerical overflow. Valid for BYTE, SHORT, LONG, FLOAT, DOUBLE and COMPLEX images.

7.7 VIP Image Statistics Functions

Histogram_Image, Image_Min_Max, Image_Mean, Image_Count, Image_Variance

#include “vip.h”

int Histogram_Image(IMAGE *image, int histogram[256]);

Function to generate histogram of grey levels in an image.

int Image_Min_Max(IMAGE *image, double *min1, double *max1, double *min2, double *max2, double *min3, double *max3);

Calculates the lower and upper bounds of pixel values of an image. The function works for all image types. Parameters min2, max2, min3, and max3 are used only for RGBTYPE and HSITYPE images. For COMPLEXTYPE images, min3 and max3 are not used. Returns ERROR (0) if unsuccessful.

int Image_Mean(IMAGE *image, double *mean1, double *mean2, double *mean3);

Calculates the mean pixel value in an image. The function works only for BYTETYPE, COMPLEXTYPE, RGBTYPE, and HSITYPE images. For BYTETYPE, mean2 and mean3 are not used; and for COMPLEXTYPE, mean3 is not used. Returns ERROR (0) if unsuccessful.

int Image_Count(IMAGE *image, int *sum1, int *sum2, int *sum3);

Is similar to function Histogram Image except that it covers RGBTYPE and HSITYPE images as well. sum1, sum2, and sum3 should be arrays of at least 256 elements. The function takes BYTETYPE, RGBTYPE, and HSITYPE images. Parameters sum2 and sum3 are not used for BYTETYPE images. Returns OK (1) if successful, ERROR (0) otherwise.

int Image_Variance(IMAGE *image, double mean1, double mean2, double mean3, double *var1, double *var2, double *var3);

Calculates the variance of pixel values of an image. Function Image Mean should be called prior invoking this function. mean1, mean2, and mean3 are the mean values calculated by Image Mean. The function works for all image types (see function below). Returns ERROR (0) if unsuccessful.

7.8 VIP Frequency Domain Functions

FFT_Shuffle, Shift_Image_Origin, Image_Complex_Conjugate, FFT, FFT_Image, IFFT_Image, FT_Image, fourn, Make_Homomorphic_Filter

#include "vip.h"

void FFT_Shuffle(int n, IMAGE *f, int offset, int no);

Reorders the complex image f whose row (or column) size is n (n must be an integer power of 2) for the computation ofIFFT, FFT Image, IFFT Image. The shuffling is done on a one-dimensional array of a two-dimensional complex image. Value stored in the last two arguments contains the following key information:

offset = 0 - shuffle on column number = no

offset = 1 - shuffle on row number = no

otherwise nothing is performed.

Note that f need not be a square image.

int Shift_Image_Origin(IMAGE *cin, IMAGE *cout);

Shift the origin of the image (default image origin is at the top-left corner) to the image's centre. Given an input image divided into four quadrants, these four quadrants will be swapped as shown in the diagram below.

```
+---+---+            +---+---+
|   |   |            |   |   |
| 1 | 2 |            | 4 | 3 |
|   |   |            |   |   |
+---+---+     ==>    +---+---+
|   |   |            |   |   |
| 3 | 4 |            | 2 | 1 |
|   |   |            |   |   |
+---+---+            +---+---+
(input)              (output)
```

Image_Complex_Conjugate(IMAGE *im);

Reverse the sign of the imaginary component of a COMPLEX image. Note that Image_Complex_Conjugate is one of the functions that does not preserve the content of images passing to it. Image Complex Conjugate can be modified in the future if necessary.

int FFT(IMAGE *image);

FFT takes in a COMPLEXTYPE image and overwrites it with its unscaled Fourier Transform. The following computation is stored to image:

```
          M-1 N-1
F(u,v) = Sum Sum { f(x,y) exp [-j*2*pi*(ux/M + vy/N)] }
          x=0 y=0
```

Note that FFT is one of the functions that does not preserve the content of images passing to it.

IMAGE *FFT_Image(IMAGE *image, int origin_shift);

FFT IMAGE takes in a COMPLEXTYPE image and returns the Fourier Transform of the input image (the origin image is not overwritten) if everything works fine. Function returns a NULL image if any one of the following conditions is not satisfied: 1. no space has been allocated for either image;
2. dimensions of the image are not integer powers of 2.

FFT Image outputs the following:

```
            1  M-1 N-1
F(u,v) = ---- Sum Sum { f(x,y) exp [-j*2*pi*(ux/M + vy/N)] }
           MN  x=0 y=0
where MxN are the image dimensions, and j power 2 = -1.
```

If the second argument, origin_shift is non-zero, Shift_Image_Origin is called prior the return of the function.

IMAGE *IFFT_Image(IMAGE *image, int origin_shift);

IFFT Image takes in a COMPLEXTYPE image and returns the Inverse Fourier Transform of the input image (the origin image is not overwritten) if everything works fine. Function returns a NULL image if any one of the following conditions is not satisfied: 1. no space has been allocated for either image;
2. dimensions of the image are not integer powers of 2.

IFFT_Image outputs the following:

```
            1  M-1 N-1
f(x,y) = ---- Sum Sum { F(u,v) exp [j*2*pi*(ux/M + vy/N)] }
           MN  x=0 y=0
where MxN are the image dimensions, and j power 2 = -1.
```

If the second argument, origin_shift is non-zero, Shift_Image_Origin is called prior the above computation.

IMAGE *FT_Image(IMAGE * inimage, int shift, char *forward_inverse);

Function to generate the Fast Fourier Transform of an image.

input: inimage - pointer to image to be transformed.

shift	- if non-zero the origin of the image is shifted so that zero frequencies are placed in the middle of the image.
forward_inverse	- “forward” or “inverse”.

returns: pointer to transformed image.

void fourn(float data[], int nn[], int ndim, int isign);

An N dimensional FFT, (from Numerical Recipes in C p468 Original code from N.M. Brenner of Lincoln Labs). This function replaces data by its ndim dimensional discrete Fourier transform, if isign = 1 nn[1..ndim] is an integer array containing the lengths of each dimension (No of complex values), which must be powers of 2.

data is a real array of length twice the product of these lengths, in which the data are stored as in multidimensional complex array: real and imaginary parts of each element in consecutive locations, and the rightmost index of the array increases most rapidly as one proceeds along data. For a 2D array this is equivalent to storing the data by rows.

If isign = -1, data is replaced by its inverse transform.

IMAGE *Make_Homomorphic_Filter(int rows, int cols, double CutOff, double LowGain, double HighGain, int order);

Function to create a Butterworth-like Homomorphic filter. It is created in the form of a FLOAT image of the specified size which can then be used to scale the FFT of an image.

Note: The filter is constructed assuming that the FFT of the image has not had its origin shifted to the centre.

No checks are made on the bounds of HighGain and LowGain but for 'sensible' results LowGain should be between 0 and 1 and HighGain should be greater than 1. A starting point would be to use a LowGain of 0.5 and a HighGain of 2.0

7.9 VIP Image Calibration Functions

xyz2uv, xyz2uvNC, uv2xyz, Read_Plane_List, Calc_Sensor, Set_Plane, Stereo_Point, Lens_Comp_4, Lens_Comp_3, Calibrate, Read_Calib_Pts, Read_Image_Pts, Write_Camera_Txt, Read_Image_Pts_From_Calib_File, Read_Camera

#include "vip.h"

void xyz2uv(int uv[2], double xyz[3], CAMERA camera);

This function converts xyz world coordinates to integer uv image coordinates. Image coordinate values are clamped to 0-511. Typically this function is used to determine where to search for a feature given an approximate position in machine coordinates. Note that the uv image coordinates are really frame-grabber coordinates so you must allow for umin and vmin (the image offset from the frame-grabber origin) when determining which row and column in the image to go to.

void xyz2uvNC(int uv[2], double xyz[3], CAMERA camera);

This function converts xyz machine coordinates to uv image coordinates This routine differs from xyzTOuv in that there is no clamping of uv values to 0-511. Used occasionally.

void uv2xyz(int uv[2], double xyz[3], SENSOR sensor);

This routine converts image uv coords to world xyz coords using a sensor structure. The sensor structure is generated from a CAMERA structure and a constraint plane. Note again that the uv image coordinates are really frame-grabber coordinates so you must add the values of umin and vmin (the image offset from the frame-grabber origin) to the image row and column values to obtain the values of u and v to pass to this function.

int Read_Plane_List(char *filename);

This function reads a plane list file and places the data in global array PLANE (PLANE[NPLANES][3][3] is defined in vipglobals.c). Each plane is defined by three points.

Expected format of a plane list file is as follows:

```
* Coments begin with '*' . Blank lines can be anywhere

plane n

 x1  y1  z1
 x2  y2  z2
 x3  y3  z3

plane n+1
etc.
etc.
```

plane numbering does not have to be in order. If a plane number is repeated the plane data is overwritten with the new plane. (This should be fixed to give you a warning if this happens).

int Calc_Sensor(CAMERA camera, double planepts[3][3], SENSOR *sensor);

input:
camera - camera calibration matrix.
planepts[3][3] - 3 points defining a plane.
output:
sensor - pointer to sensor matrix.

Routine to set up a sensor matrix. The sensor matrix is the transformation between image uv coords and machine xyz coords. It is generated from the camera calibration matrix and the coordinates of three points defining the plane in which the measurements are assumed to lie. The No of the camera that is currently being used is also set. See Bolles, Kremers and Cain "A Simple Sensor to Gather Three-Dimensional Data", SRI Technical note 249 July 1981, for the derivation of the equations.

int Set_Plane(int planeNo, double *data);

Function to allow planes to be specified by user or from other vision measurements rather than from the plane list file.

int Stereo_Point(double uv1[2], CAMERA cam1, double uv2[2], CAMERA cam2, double P[3]);

input:
uv1[2], uv2[2] - image uv coords.
cam1, cam2 - cameras associated with uv coords.
output:
P[3] - world xyz coord.

The function calculates world xyz position of a point given the uv coordinates from two cameras.

int Lens_Comp_4(int iuv[ ], double duv[ ], CAMERA camera, int npts);

input:
iuv[ ] - integer pixel coords.
camera - CAMERA data structure.

output:
duv[ ] - real valued compensated coordinates.

4th order Function to compensate for lens distortion. Currently not used as fourth order polynomials were found to be too unstable.

int Lens_Comp_3(int iuv[ ], double duv[ ], CAMERA camera, int npts);

input:
iuv[ ] - integer pixel coords.
camera - CAMERA data structure. output:
duv[ ] - real valued compensated coordinates.

This is a 3rd order Function to compensate for lens distortion. This function is better to use than Lens_Comp_4 but is currently not 'connected' to the software. Normally this is called from within uv2xyz and the user need not be concerned with its use.

int Calibrate(double uv[ ][2], double xyz[ ][3], int npts, CAMERA camera, char *filename, int lenscomp);

input:

uv[ ][2]	- observed image positions.
xyz[ ][3]	- 3D positions of calibration points.
npts	- No of points.
filename	- input: Filename to place calibration data.
lenscomp	- input: Flag (1/0) indicating whether or not lens distortion compensation should be calculated.

output:

camera	- Camera data structure.

This is a camera calibration function. See Bolles, Kremers and Cain "A Simple Sensor to Gather Three-Dimensional Data", SRI Technical note 249 July 1981, for the derivation of the equations. A third order polynomial lens distortion compensation function is also calculated. This 3nd order polynomial correction function (hopefully) corrects for lens non-linearity. See Bowman and Forrest "Transformation calibration of a camera mounted on a robot", Image and Vision Computing Vol 5, No 4 November 87. pp 261-266.

Inputs are:
An array of image coordinates (actually frame grabber coordinates).
An array of corresponding world xyz coordinates.
The number of calibration points.
The name of the output file.
A flag (1/0) indicating whether or not a lens distortion compensation
function should be generated.
Output is a CAMERA structure containing a calibration matrix
and lens distortion factors.

int Read_Calib_Pts(char *filename, double CalibPts[ ][3], int *Npts);

Function to read a calibration data file containing 3D coordinates of calibration points.

Expected file format:

```
File description, comments etc
```

```
point 1 x y z
point 2 x y z
  .   . . . .
point n x y z
```

All lines in the file are ignored unless they start with the word 'POINT'. Hence, text description can be used liberally. Point numbers are ignored and are simply there for convenient reference.

int Read_Image_Pts(char *filename, double ImagePts[ ][2], int *Npts);

Function to read a calibration data file containing image data points.

Expected file format:

```
File description, comments etc

imagepoint 1 u v
imagepoint 2 u v
  .    . . . .
imagepoint n u v
```

All lines in the file are ignored unless they start with the word 'IMAGEPOINT'. Hence, text description can be used liberally. Point numbers are ignored and are simply there for convenient reference.

int Write_Camera_Txt(char *filename, CAMERA camera);

Function to writes a CAMERA data structure into a text file.

int Read_Image_Pts_From_Calib_File(char *filename, double uv[ ][2], int *Npts);

This function reads image uv coords from a text camera calibration data file into a an N x 2 array of doubles. This function is useful when one is experimenting with new calibration procedures on existing calibration data.

CAMERA *Read_Camera(char *filename);

This function to reads a binary camera data file allocates space for a CAMERA structure and returns a pointer to it.

There are two data structures involved in these transformations, the CAMERA structure and the SENSOR structure. The CAMERA structure contains all the camera calibration data. The camera calibration matrix describes the mapping of 3D world coordinates to image coordinates. The flag "calibstatus" can be: UNCALIBRATED, MATRIX_ONLY, or MATRIX_PLUS_POLY.

CAMERA structure:

```
typedef struct {
    int      idcode;
    char     version[8];
    int      channel;        /* Channel on PIP board */
    int      gain, offset;   /* Gain and offset for the
                                channel */
    int      uvoffset;       /* UV offset in using
                                calibration matrix and
                                compensation data  */
    int      calibstatus;    /* Flag indicating type of calibration
                                data present */
    double   gamma[5];       /* For grey scale correction
                                if needed */
    double   calib_matrix[3][4];/* Calibration matrix */
    double   Ucomp[18];      /* Coeffs of lens compensation
                                polynomial */
    double   Vcomp[18];      /* Coeffs of lens compensation
                                polynomial */

}  CAMERA;
```

The SENSOR structure is used to convert image coordinated to 3D world coordinates. It is generated from a camera calibration matrix and the equation of a constraint plane in which it is assumed measured points lie.

SENSOR structure:

```
typedef struct {
   double matrix[4][3]; /* The matrix that does the conversion */
   CAMERA camera;       /* Camera data for uvoffset and lens
                           distortion data */

}  SENSOR;
```

7.10 VIP Vector and Matrix Functions

Max_i, Max_l, Max_f, Max_d, Min_i, Min_l, Min_f, Min_d, Limit_Value_i, Limit_Value_l, Limit_Value_f, Limit_Value_d, Out_Of_Range_i, Out_Of_Range_l, Out_Of_Range_f, Out_Of_Range_d, Move_i, Move_f, Move_d, Movexi, Movexf, Movexd, Add_i, Add_d, Sub_i, Sub_d, Piv_i, Piv_d, Scal_i, Scal_d, Dot_i, Dot_d, Crosp_i, Crosp_d, Norm_f, Norm_d, Mag_f, Mag_d, Copy_Int_To_Double, Copy_Double_To_Int, Allocate_Matrix, Free_Matrix, Mul_i, Mul_d, Transpose_i, Transpose_d, Homog_Trans, Gauss, GaussX, Gauss_Cond, Print_Matrix_d

#include "vip.h"

Several versions of each function exist depending on the data type of the arguments. The convention is: if the function expects integers it will have postscript i, long integers - postscript l, floats - f and doubles - d. Note that C does all its floating point operations in double precision. Hence only integer and double versions of functions will typically be supported.

The general form of list of arguments for many of the vector manipulation functions are:

pointer to vector 1, size of incremental step along vector 1,
pointer to vector 2, size of incremental step along vector 2, etc.
number of elements to which these operations must be performed.

For example, Add_i(int *v1, int i1, int *v2, int i2, int *v3, int i3, int n);
int *v1, *v2, *v3;

- add the contents of v2 to v1 and place the result in v3 stepping along each vector at the specified increments for n elements.

Usually the increment size is one, however for matrix operations other increments may be useful. For example if one wanted to rescale the first column of a 3x3 matrix of integers,

```
| a1  a2  a3 |
| a4  a5  a6 |  one would use Scale_i(scalar, a, 3, a, 3, 3)
| a7  a8  a9 |
```

Note that v1, v2 etc can be the same vector, also increments may be zero or negative.

Eg. to initialize an integer vector v1 of n elements to zero one could use:
zero = 0:

Move_i(&zero, 0, v1, 1, n);

(Note how the address of variable 'zero' was used.)

To reverse the order of an array one can use:

Move_i(&v1[n-1], -1, v1, 1, n);

Functions returning the largest element in an array

int Max_i(int array[ ], int n, int *element);

long Max_l(long array[ ], int n, int *element);

float Max_f(float array[ ], int n, int *element);

double Max_d(double array[ ], int n, int *element);

Functions returning the smallest element in an array

int Min_i(int array[ ], int n, int *element);

long Min_l(long array[ ], int n, int *element);

float Min_f(float array[ ], int n, int *element);

double Min_d(double array[ ], int n, int *element);

Functions to clamp a value to limits

Functions return 1 if variable had to be clamped, 0 if not.

int Limit_Value_i(int *x, int xmin, int xmax, char *error_message);

int Limit_Value_l(long *x, long xmin, long xmax, char *error_message);

int Limit_Value_f(float *x, float xmin, float xmax, char *error_message);

int Limit_Value_d(double *x, double xmin, double xmax, char *error_message);

Functions to test whether an variable is out of limits

Functions return 1 if out of range, 0 if not.

int Out_Of_Range_i(int x, int xmin, int xmax, char *message);

int Out_Of_Range_l(long x, long xmin, long xmax, char *message);

int Out_Of_Range_f(float x, float xmin, float xmax, char *message);

int Out_Of_Range_d(double x, double xmin, double xmax, char *message);

Functions to move values from one array to another

Arrays may overlap.

int Move_i(int *v1, int i1, int *v2, int i2, int n);

int Move_f(float *v1, int i1, float *v2, int i2, int n);

int Move_d(double *v1, int i1, double *v2, int i2, int n);

Functions to move values from one array to another

Arrays v1 and v2 must not overlap.

int Movexi(int *v1, int i1, int *v2, int i2, int n);

int Movexf(float *v1, int i1, float *v2, int i2, int n);

int Movexd(double *v1, int i1, double *v2, int i2, int n);

Vector Add Functions v3 = v1 + v2

int Add_i(int *v1, int i1, int *v2, int i2, int *v3, int i3, int n);

int Add_d(double *v1, int i1, double *v2, int i2, double *v3, int i3, int n);

Vector Subtract Functions v3 = v1 - v2

int Sub_i(int *v1, int i1, int *v2, int i2, int *v3, int i3, int n);

int Sub_d(double *v1, int i1, double *v2, int i2, double *v3, int i3, int n);

Vector Pivot Functions v3 = scal*v1 + v2

int Piv_i(intscal, int *v1, int i1, int *v2, int i2, int *v3, int i3, int n);

int Piv_d(scal, double *v1, int i1, double *v2, int i2, double *v3, int i3, int n);

Vector Scale Functions v2 = scal*v1

int Scal_i(int scal, int *v1, int i1, int *v2, int i2, int n);

int Scal_d(double scal, double *v1, int i1, double *v2, int i2, int n);

Dot Product Functions. Dot product of v1 and v2 returned

int Dot_i(int *v1, int i1, int *v2, int i2, int n);

double Dot_d(double *v1, int i1, double *v2, int i2, int n);

Cross Product Functions v3 = v1 x v2

int Crosp_i(int v1[ ], int v2[ ], int v3[ ]);

int Crosp_d(double v1[ ], double v2[ ], double v3[ ]);

Functions to normalize vectors

Functions normalize a vector and return its original magnitude.

float Norm_f(float array[ ], int n);

double Norm_d(double array[ ], int n);

Functions to find the magnitude of vectors

float Mag_f(float array[ ], int n);

double Mag_d(double array[ ], int n);

Conversion between arrays of ints and doubles

void Copy_Int_To_Double(int *v1, int i1, double *v2, int i2, int n);

Copy_Double_To_Int(double *v1, int i1, int *v2, int i2, int n);

Function to allocate memory for a matrix

char **Allocate_Matrix(int rows, int cols, int size);

This function dynamically allocates space for a 2D matrix in such a way that one can address individual elements in the usual way. It does this by allocating a contiguous section of memory for the complete 2D array and then allocating an array of pointers of size rows, each element being assigned the address of the element in the main array corresponding to the start of each row. This means that you should NOT pass the name of the allocated matrix as an address of the matrix to functions as you will only be passing the address of the array of pointers to the rows. You should use matrix[0] or &matrix[0][0] as in the example below.

Example of use:

```
double **matrix;
double one = 1.0, three = 3.0;

matrix = (double **)Allocate_Matrix(3, 6, sizeof(double));

Move_d(&one,0,&matrix[0][0],6,3)  (set 1st column to 1's)
Move_d(&three,0,&matrix[0][2],6,3) (set 3rd column to 3's)
matrix[0][3] = 1.0;  (etc)
```

Function to free a matrix

int Free_Matrix(char **matrix);

Matrix Multiplication Functions: C = A * B

int Mul_i(int l, int m, int n, int *A, int *B, int *C);

l - rows in A.
m - columns in A.
n - columns in B.

int Mul_d(int l, int m, int n, double *A, double *B, double *C);

l - rows in A.
m - columns in A.
n - columns in B.

Matrix Transpose Functions

Transpose_i(int l, int m, int *A, int *AT);

l - rows in A.
m - columns in A.

int Transpose_d(int l, int m, double *A, double *AT);

l - rows in A.

m - columns in A.

Homogeneous Transformation Function

int Homog_Trans(double A[4][4], double b[3], double c[3]);

double A[4][4], b[3], c[3];

Gaussian Elimination Functions (for doubles)

Solves x for Ax = b partial pivotal condensation is used. Code from Dorn and McCracken "Numerical Methods in Fortran IV" translated into C by PK.

int Gauss(double A[ ], double x[ ], double b[ ], int n);

int GaussX(double Abuf[ ], double x[ ], double b[ ], int m, int n);

double Gauss_Cond(double Abuf[ ], double x[ ], double b[ ], int m, int n);

Also returns condition number.

Matrix Printing Function (for doubles)

void Print_Matrix_d(double *M, int rows, int cols);

7.11 VIP Image 1D Edge Detector Functions

Edge_1D, MEdge_1D, Convolve_Array_i, *SD_Gauss_Mask_1D, D_Gauss_Mask_1D, Gauss_Mask_1D, Find_1st_Peak, Find_Closest_Peak, Select_Peaks

#include “vip.h’

int Edge_1D(IMAGE *image, int ustart, int vstart, int uend, int vend, INT_MASK mask, int lookup[256], int updown, int span, int threshold, int *cu, int *cv);

A one dimensional edge detection function.

lookup - lookup table to modify intensity values.
updown - +ve for upward transition -ve for downward.

int MEdge_1D(IMAGE *image, int uend1, int vend1, int uend2, int vend2, int width, int ntimes, INT_MASK mask, int lookup[256], int updown, int span, int threshold, int cu[ ], int cv[ ]);

Multiple line search for an edge.

lookup - lookup table to modify intensity values.

Convolve_Array_i(int array[ ], int arraysize, INT_MASK mask);

Function to convolve a 1D array of ints with a mask.

INT_MASK *SD_Gauss_Mask_1D(int isigma);

Function to set up 2nd derivative of a Gaussian mask. Note that a negative mask is generated (ie it goes down then up rather than vice versa) so that upward intensity peaks are marked by upward peaks in the convolution result.

INT_MASK *D_Gauss_Mask_1D(int isigma);

Function to set up 1st derivative of a Gaussian mask. Note that a negative mask is generated (ie it goes down then up rather than vice versa) so that upward intensity gradients are marked by upward peaks in the convolution result.

INT_MASK *Gauss_Mask_1D(int isigma);

Function to set up a Gaussian mask.

int Find_1st_Peak(int *buffer, int Npts, int UpDown, int span, int threshold);

Function to find the first peak in an array of integers. The flag 'UpDown' is used to specify a search for a peak or a trough. If we are looking for a trough the values are multiplied by -1 so that only a +ve peak has to be found. Only peaks above 'threshold' are flagged. If no peaks above threshold are found ERROR is returned. It is assumed that the first point is not the peak.

int Find_Closest_Peak(int *buffer, int Npts, int UpDown, int span, int threshold, int position);

Function to find the peak in an array of integers that is closest to a specified position in the array. The flag 'UpDown' is used to specify a search for a peak or a trough. If we are looking for a trough the values are multiplied by -1 so that only a +ve peak has to be found. Only peaks above 'threshold' are flagged. If no peaks above threshold are found ERROR is returned. It is assumed that the first point is not the peak.

int Select_Peaks(int *buffer, int Npts, int FirstPeakWanted, int LastPeakWanted, int UpDown, int Direction, int span, int threshold, int *peak, int *NpeaksFound);

Function to select a number of peaks in an array of integers. The search can be specified from either end of the array. The flag 'UpDown' is used to specify a search for a peak or a trough. If we are looking for a trough the values are multiplied by -1 so that only a +ve peak has to be found. Only peaks above 'threshold' are flagged. If no peaks above threshold are found ERROR is returned. It is assumed that the first point is not the peak.

7.12 Functions for Complex Arithmetic

Complex_Mult, Complex_Div, Complex_Sub, Complex_Add, Complex_Magnitude, Init_Complex_Vector, Allocate_Complex_Vector, Free_Complex_Vector

#include "vip.h"

void Complex_Mult(z1, z2, z3)

COMPLEX *z1, *z2, *z3;

Complex_Mult multiplies two complex numbers together. The result is stored in the third parameter (z3 = z1 * z2).

void Complex_Div(z1, z2, z3)

COMPLEX *z1, *z2, *z3;

Complex_Div divides a complex number by another complex number. The result is stored in the third parameter (z3 = z1 / z2).

void Complex_Sub(z1, z2, z3)

COMPLEX *z1, *z2, *z3;

Complex_Sub subtracts a complex number from another complex number. The result is stored in the third parameter (z3 = z1 - z2).

void Complex_Add(z1, z2, z3)

COMPLEX *z1, *z2, *z3;

Complex_Add sums two complex numbers (z3 = z1 + z2).

float Complex_Magnitude(z)

COMPLEX *z;

Complex_Magnitude returns the magnitude of a complex number.

void Init_Complex_Vector(v, n)

COMPLEX *v;

Init_Complex_Vector initializes all the elements in an array of complex structure of size n to zeroes.

COMPLEX *Allocate_Complex_Vector(nc)

int nc;

Allocate_Complex_Vector allocates space for an array of n elements of complex structure. The function returns the address of the allocated array or NULL if unsuccessful.

int Free_Complex_Vector(v)

COMPLEX *v;

Free_Complex_Vector frees the space previously allocated to an array of complex structure. Returns 1 if the deallocation is successful, 0 otherwise.

7.13 VIP Miscellaneous Functions

VIP_Error_Msg, Image_OK, Common_Window, Pixel_In_Image, Find_Spot, Wind_Int, Closest_Power_Of_2, Is_2_Power, Power_Of_2_Greater_Or_Equal,

#include "vip.h"

void VIP_Error_Msg(char *message);

VIP_Error_Msg is a function to handle error messages. Error messages are accumulated in the global string variable VIP_MESSAGE so that causes of errors can be traced back through whatever path functions were called. Error messages are also printed out to VIP_STD_ERR and written to the log file (if it is open). Programs written for XView may not want error messages printed out to the screen, but rather displayed in alert boxes. This can be achieved by setting VIP_STD_ERR to NULL and passing the contents of VIP_MESSAGE to the alert box displaying procedure. Remember to clear the contents of VIP_MESSAGE after displaying it.

int Image_OK(IMAGE *image, char *message);

Image_OK tests the validity of the pointer to an image structure. It checks that the pointer is not NULL, checks that the idcode is correct for an image and checks that the memory sentinels at the start and end of the image data have not been corrupted. All VIP functions use this function to test the validity of any images that have been passed to them.

int Common_Window(IMAGE *image1, IMAGE *image2, int umin, int vmin, int rows, int cols);

Common_Window determines the common area between two images.

int Pixel_In_Image(IMAGE *image, int u, int v);

Pixel_In_Image tests whether pixel (frame grabber) coordinates are within image bounds. Returns TRUE or FALSE.

int Find_Spot(IMAGE *image, int umin, int vmin, int umax, int vmax, int thresh, int bw, int *cu, int *cv);

Function to find the centroid of a blob within a window in an image. Returns No of spots found (0 or 1).
bw - 1 for white blob on black, bw = 0 for black on white.
cu, *cv - coords of blob centre.

int Wind_Int(IMAGE *image, int umin, int vmin, int umax, int vmax, int *avint);

umin, vmin, umax, vmax - search area.
avint - average intensity.

Function to calculate average intensity in a window.

int Power_Of_2_Greater_Or_Equal(int x);

Returns the greater or equal power of two for the given number.

int Closest_Power_Of_2(int x);

Returns the closest power of two for the given number.

int Is_2_Power(int n, int *logn);

Returns 1 if the first argument, n, is an integer power of 2, 0 otherwise. Also return a number, logn, to the second argument where 2 power logn = n.

Index

Program and function index