IKS: Image Kernel System Users' Manual

Ashok Samal and Tom Henderson

UUCS-87-020

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

August 17, 1987

Abstract

IKS (for *I*mage *K*ernel *S*ystem), is an image processing system, currently being used by the Computer Vision and Image Processing group at the University of Utah. IKS provides a convenient environment for reading, writing, and manipulating images. The system consists of two types of routines. First, there are functions which can be used in C programs to manipulate images. There are also image operators which work on images directly. They can be called from unix like system utilities.

This report describes the components and capabilities of IKS. It also describes how to use the system on various machines in the department on which it is available.

Contents

1 Introduction

IKS (for *I*mage *K*ernel *S*ystem), is an image processing system, currently being used by the Computer Vision and Image Processing group at the University of Utah. It is based on a vision system developed at the University of British Columbia. The original system has undergone several changes to suit local needs and lot of utility functions are also added.

IKS provides a convenient environment for reading, writing, and manipulating images. The system consists of two types of routines. First, there are functions which can be used in C programs to manipulate images, e.g., open and close image files. There are also image operators which work on images directly. They can be called from unix like system utilities. For example, the *transpose* filter takes an image as input and produces an image which is the transpose of the input image. There are several such filters in IKS.

2 IKS Image File Format

Images are represented by a rectangular array of packed pixel values. Additionally, the file contains an image file descriptor called the *header*. The header contains a set of image parameters common to most images and an annotation field for additional user-defined parameters. There is also an optional field for a textual description of the image. The common image parameters are:

- **nrows** : The number of rows in the image.
- **ncols** : The number of columns in the image.
- **bpp** : The number of bits per pixel required to represent the full range of intensity values for the image.
- **signed** : Boolean value indicating whether the integer pixel values are represented in pure magnitude form or signed 2's complement format.
- **positive** : Boolean value indicating the image was produced from a photographic positive or negative source.

Most of the parameters have default values and hence, may be ignored when not appropriate. The maximum number of bits per pixels 32. The user parameters in the annotation fields can be accessed via *sscanf(3)* and *sprintf(3)*. The definition of the IMAGE structure is given in Appendix A.

An image may require a very large amount of space. For example, a 1K by 1K image, with 8 bits per pixel, needs 1 MBytes. To store such large data objects efficiently in the file system of the host operating system, the input/output system is implemented as bit streams, called *bitio*. Essentially, the file is treated as randomly accessible accessible contiguous streams of bits and the byte boundaries are ignored.

3 Image File Manipulation Primitives

IKS provides several primitive functions to manipulate image files encoded in the standard image file format described in the previous section. As already mentioned the image file contains a header filed followed by the pixels of the image and are accessed via bit stream I/O. They include functions to open and close image files, read and write headers into the image files, access the contents of the images, etc. The complete list of primitive functions are given in Appendix B. Also, an example showing the usage of these routines is given in Appendix D.

While the list of functions provided is rather small, they represent the core of any image processing system and more complicated routines can be built on top of them.

4 Direct Image Transformations

The primitive functions described in the previous section help in developing programs to manipulate image from scratch. However, often one may want to perform a high level operation on an image, e.g., histogramming, edge detection. IKS provides many such operators, which directly operate on the image and the user does not have to do any programming.

These operators can be grouped into three categories. First, there are operators which are very commonly used. Typical examples are windowing, masking, convolution, etc. There are also several other image processing operators, which are not commonly used, but are useful, in the sense that they make the system easier to use. For example, there is a program which lets the user edit the image headers. There are also I/O programs to handle the images in the IKS format. All the image operators are listed in alphabetic order in Appendix C.

It is important to note that this list is undergoing continuous change. More and more operators are being added to the list. For an up to date list, you should look into the appropriate directories in the system. It is also important to note that many of these programs can be connected to each other and to other Unix utilities by pipes. See Appendix E for an example.

5 Documentation

This manual describes the basic philosophy of the IKS system, its capabilities, and its usage. Extensive documentation on various aspects of the system is kept on-line similar to the Unix man pages. After setting up the right environment, they can be accessed by the usual Unix *man* command.

If you want to know more about the system, the best way is to browse around the source directories. They are publicly readable and are kept in "$(IKS)/src". There are "README" files in every directory explaining what the directory contains.

6 Pragmatics

It is very easy to start using the IKS system. Currently IKS is available on two machines: *ug.utah.edu* and *sp.utah.edu*. Several things must be done to access the IKS system functions and libraries. They are briefly described below:

1. Add $(IKS)/bin to your search path in your .login/.cshrc file. All the image operators are in this directory.

2. Include $(IKS)/include/image.h file in order to access the IKS primitive functions from a C program. This may be done explicitly in your source code or by using the "-I" option in a makefile (see Appendix F). You also need to load the library "$(IKS)/lib/image.a" in order to access these functions.

3. Set the environment variable MANPATH to access the IKS man pages, i.e., insert "*setenv* MANPATH /usr/man:/usr/local/man/$(IKS)/man" in your .login/.cshrc file.

$(IKS) is "/v/misc/vision/IKS" in ug.utah.edu and "/s/IKS" in sp.utah.edu.

7 Contributing to IKS

The system is being continually updated, mainly to add new functionalities to the system. The user are encouraged to contribute to the system. If you want your code to be a part of IKS, there are some things you should keep in mind. Write your code so that it can take input from *stdin* and write into *stdout*. This way, it can be used in conjunction with other image operators available in the system. You are also expected to write the documentation for your operator similar to Unix man pages. A skeleton man page is kept in "$(IKS)/examples/skel-man.l" for your reference.

To install your code, copy your source code into "$(IKS)/stage". Copy the man page into "$(IKS)/stage/man" directory. If you have more than one source file make your own directory and put the code there. If you are using makefile, copy that into the staging area as well. Please check to make sure that you don't overwrite any files. *Also, make sure that the files are readable and writable by everybody.* This is a publicly writable area and after the code is installed, these files are removed. All these instructions are kept in "$(IKS)/stage/install.inst" file. Finally send mail to "samal@cs.utah.edu" explaining about your code, etc.

8 Reporting Bugs

The system is mostly bug-free. However, if you come across bugs, report the exact scenario when the bug appeared to "samal@cs.utah.edu". The bugs will be fixed as soon as possible.

If you have any other questions send mail to the above address as well.

Appendix A

Image File Definitions

```
#define IMVERSION 2          /* iff version number                  */
#define IMBPP     8          /* Default number of bits per pixel    */
#define IMSIGNED  0          /* Default image is unsigned           */
#define IMPOSTVE  1          /* Default image is positive encoded   */

#define IMBLKSZE BBLKSZE     /* Use block size of bitio file (4096) */
#define IMHDRSZE 1024        /* Physical header size in file        */

typedef struct image
       {
        BFILE   *bfile;      /* bitio file pointer                  */
        long    nrows;       /* Number of rows in image             */
        long    ncols;       /* Number of columns in image          */
        short   bpp;         /* Bits per pixel                      */
        short   signed;      /* Signed / Unsigned image             */
        short   positive;    /* Positive / Negative image           */
        short   hdroffset;   /* Header offset in file               */
  } IMAGE;
```

Appendix B

Primitive Functions in IKS

- **iopen** (name, mode) char *name; short mode;
  Opens the file *name* as an image file. Returns an image file pointer of type IMAGE containing only default values for the image parameters (see getheader below). The mode argument is one of 0 (read), 1 (write), or 2 (read/write). On error it returns a NULL.

- **idopen** (fd, mode) int fd; short mode;
  Associates an image file with the file descriptor fd for an open Unix file. Otherwise behaves as **iopen**above. On error it returns a NULL.

- **getheader** (ifp, annote) IMAGE *ifp; char annote[IMHDRSZE];
  Reads the header of the image file (previously opened with **iopen** or **idopen**) into the image file descriptor ifp of type IMAGE. Copies the annotation record of ifp into the string annote appending trailing nulls. The size of annote must be at least IMHDRSZE. The mode of the file must be either 0 (read) or 2 (read-write). Default image parameters are thus over written in ifp by the actual parameters of the image. **getheader** returns 0 on success and -1 on failure. **getheader** should be called before pixels are read from or written to the file. The file pointer is left pointing to pixel[0,0].

- **putheader** (ifp, annote) IMAGE *ifp; char annote[];
  Writes the standard image parameters and the annotation to the image file. Annote may be of length 0 to IMHDRSZE less the length of the standard parameter field. Annote is truncated or padded with nulls as necessary to insure a final header size of length IMHDRSZE. The mode of the file must be either 1 (write) or 2 (read-write). **putheader** returns 0 on success and -1 on failure. **putheader** should be called before any pixels are written to the file. The file pointer is left pointing to pixel[0,0].

- **iclose** (ifp) IMAGE *ifp;
  Closes the image file flushing any unwritten pixels. Returns 0 on success and -1 on failure.

- *long* **getpix** (ifp) IMAGE *ifp;
  Returns the next pixel from the image file in raster order. On EOF, value 0L is returned.

- **putpix** (ifp, pixel) IMAGE *ifp; long pixel;
  Puts the value of pixel into the next pixel of the image file in raster order.

- **getrow** (ifp, row, buf) IMAGE *ifp; int row; long buf[];
  Reads a row of pixels from image *ifp into array buf beginning at row row. buf must be type long (signed or unsigned) and big enough to hold an entire row of the image.

- **putrow** (ifp, row, buf) IMAGE *ifp; int row; long buf[];
  Writes a row of pixels into image *ifp from array buf beginning at row row. buf must be type long (signed or unsigned) and big enough to hold an entire row of the image.

- *long* **iseek** (ifp, row, col) IMAGE *ifp; int row, col;
  Sets the file pointer in the image file to point at pixel[row,col]. Returns the absolute pixel bit location in the file on success and -1 on failure. Rows and columns (row and col values) begin at pixel[0,0].

- **isync** (ifp) IMAGE *ifp;
  Flushes the pixel buffer associated with file ifp to synchronize the file with its I/O.

- *long* **iget** (ifp, parm) IMAGE *ifp; int parm;
  Returns the value of the standard parameter denoted by parm from the image. Access to image parameters in header are accomplished by specifying an integer index corresponding to each parameter. The correspondence is as follows:

1	=	nrows
2	=	ncols
3	=	bpp
4	=	signed
5	=	positive

- *long* **iset** (ifp, parm, value) IMAGE *ifp; int parm, value;
  Sets the value of the standard parameter of the image specified by parm. See **iget** for the indexes for the parameters.

- **ieof** (ifp) IMAGE *ifp;
  Returns 1 if EOF has occurred during the last getpix or getrow and 0 otherwise.

- **ierror** (ifp) IMAGE *ifp;
  Returns 0 if no errors have occurred during last file access and some error flags otherwise.

Appendix C

Image Operators in IKS

- **afm :** Apply function memory (lookup table) to an image file.
- **autothreshold :** Threshold a bi-modal image.
- **average :** Calculate the average value at each pixel of the image.
- **cocur :** Compute the co-occurence matrix.
- **contab :** Construct a confusion table.
- **convolve :** Convolves an image with a mask.
- **display :** Display IKS images.
- **graphictext :** Produce image files of text.
- **histmatch :** Modifies the input image to fit a histogram.
- **histogram :** Computes the histogram and cumulative histogram.
- **iff-fft :** Perform direct/inverse DFT.
- **iff2mt :** Write IKS image files out onto magnetic tape.
- **iffcorr :** Calculate the correlation coefficient between two images.
- **iffedit :** Edit the standard image file format file headers.
- **iffimp :** Convert an image to an IMPRESS bitmap.
- **ifflop :** Perform logical operations on the image.
- **iffsee :** Display headers of standard IKS images.
- **iksps :** Convert IKS image file into Postscript format.
- **image :** Convert file from raw pixel raster to IKS format.
- **imask :** Overlay two images.
- **iscale :** Scale an image.
- **istat :** Gather image statistics.
- **iwindow :** Produce a sub-image of the original image.

- **median :** Calculate the median value at each pixel
- **mt2iff :** Read IKS image files off of a magnetic tape.
- **mask :** Make a mask for convolution.
- **mlc :** Maximum-likelihood image classification.
- **munch :** Reduce an image file according to a reduction count.
- **nnc :** Nearest-neighbor image classification.
- **nuxi :** Reverse the order of bytes in an image.
- **orient :** Change (rotate, flip) the orientation of an image.
- **randomdot :** Produce random dot stereograms.
- **raster :** Convert IKS image to raw pixel raster.
- **slice :** Density slice an input image.
- **splice :** Combine several small images into one large image.
- **stereo :** Produce orthographic stereo pairs.
- **synthetic :** Make a synthetic image from a digital terrain model.
- **threshold :** Threshold an image at a fixed gray level.
- **transpose :** Transpose an image.
- **xflip :** Invert the image (left-right).
- **yflip :** Invert the image (top-bottom).
- **zoom :** Enlarge an image.

Appendix D

Using IKS functions in C

The following program uses most of the basic image access functions.

```
#include <image.h>

IMAGE *im;
char annote[1024];

main()
{
    int i, j;
    long val;

    im = iopen("image1", 1);     /* Open image for writing */
    im->nrows = 10;              /* Set rows, columns and bpp */
    im->ncols = 10;
    im->bpp = 16;
    strcpy(annote,"Myimage");    /* Make own field */
    putheader(im,annote);        /* Add to header */
    for (i=0; i<10*10; i++)        /* Write pixel values */
        putpix(im,i);
    iclose(im);                  /* Close image file */

    im = iopen("image1", 2);     /* Open for Read/Write */
    getheader(im, annote);       /* Get header info */
                                 /* Access various fields */
    printf("Number of Rows    = %d\n", im->nrows);
    printf("Number of Columns = %d\n", im->ncols);
    printf("Bits per pixel    = %d\n", im->bpp);
    printf("Header : %s\n", annote);

    printf("Enter row and column ( <10 ) :");
    scanf("%d%d",&i,&j);         /* Read row and column numbers */
    iseek(im,i,j);               /* position appropriately */
    val = getpix(im);            /* Get pixel value at i, j */
    printf("Intensity at pixel %d %d is %d\n", i,j,val);

    iclose(im);                 /* Close image file */
}
```

The comments in the code explain action at each step. The program is also kept in $(IKS)/examples/demo.c. One thing that must be noted here. Before reading from, or writing into the image file, make sure that you have accessed the header.

Appendix E

Using the Image Operators

The image operators are used similar to Unix utilities. The man page for the operator explains its usage: its arguments, and the list of optional flags. For example,

istat image1

will give statistics about the image.

istat -z image1

will give the statistics along with the number of pixels with zero intensity.

If the output of the operation is an iff file then it can be piped in as the input of another operator. For example it iwindow operator creates a sub image from an image. This can be fed in as the input to another operator, say *istat*. Thus,

iwindow image100 | istat -z

will give the statistics of the subimage created by the window operator. Pipes can be made arbitrarily long as long as the input of one stage is compatible with the output of the previous stage.

Appendix F

Makefile

When using the IKS primitive functions from a C program, one must include the appropriate header files and also link in the necessary libraries. One may include the complete path name in the source code, but it makes the program less portable. Use of an appropriate "makefile" helps in this respect. A sample makefile is given here, for reference. This is also kept in "$(IKS)/examples/makefile".

```
# This is a sample makefile
ROOT = /s/IKS
INCLUDE = $(ROOT)/include
IMLIB  = $(ROOT)/lib/image.a
# the libraries
LIBS =   -lm -ltermcap $(IMLIB)

IFLAGS = -I$(INCLUDE)
CFLAGS = -g

# source files
CFILES = \
 test.c

# object files
OFILES = \
 test.o

# header files
HFILES = \
         image.h\
         bitio.h

test: $(CFILES)
cc -o test $(IFLAGS) $(CFLAGS) $(CFILES) $(LIBS)

print:
ln $(HFILES) $(CFILES)

# This entry removes all .o files
clean:
-rm *.o
```