FLIP
Floating Point Image Processing Library
This library assumes input in the form of a floating point ifs image. Outputs will be floating point images as well. The library is designed to optimize computation speed, while providing the programmer with easy-to-use tools. **If all the inputs are not float, the function returns -1. If the input dimensions are not compatible, the function returns -2.** ***Note that no error messages are printed, so the user must check the value returned***.

fladds.c

Add a scalar to an image

int fladds(img1,img2,scalar)
    IFSIMG img1,img2;
    float scalar;
Input image is img1, output image is img2, third argument is the scalar. This value is added to each pixel of img1, and the result stored in img2.

fladdv.c

Add two images
int fladdv(img1,img2,img3)
    IFSIMG img1,img2,img3;
The first input image is img1, the second input image is img2, the output is img3. Corresponding pixels of img1 and img2 are added and the result stored in img3.

flclip.c

Limit permitted brightness in images
int flclip(img1,img2,scalar)
    IFSIMG img1,img2;
    float scalar;
The first image, img1 is clipped so that its maximum value is scalar.

flcp.c

Copy an image
nt flcp(in_img,out_img)

IFSIMG in_img, out_img;
The image in_img will be copied to the image out_img

fldivs.c

Divide by a scalar
int fldivs(img1,img2,scalar)
IFSIMG img1,img2;
float scalar;
Each pixel of img1 will be divided by the scalar, and the result stored in the corresponding pixel if img.

fldivv.c

Divide two images
int fldivv(img1,img2,img3)
IFSIMG img1,img2,img3;
Each pixel of img1 is divided by the corresponding pixel of img2, and the result stored in img3. Note, no check for divide by zero is performed, so the special floating point value INFINITY can occur.

Derivative operators

All the derivative operators listed below support 1-D (signals) 2-D (images) aned 3-D (volumes) data sets. In all cases, x denotes the column and y the row.

fldx.c

Extimate partial deriviative in the x direction.
int fldx(img1,img2)
IFSIMG img1,img2;
Compute an approximation of the first deriviative with respect to x by the difference between pixels in an image images. For each pixel,
outrow[j][i] = (inrow[j][i+1] - inrow[j][i-1])*.5

fldx_back.c

Backward difference

The forward and backward differences both estimate the first partial derivative with respect to x, but do it with higher resolution (and higher noise sensitivity) than fldx. They are useful in iterative algorithms where you do one iteration using the forward difference, and the next using the backward.

int fldx_back(img1,img2)
    IFSIMG img1,img2;

Compute the backward difference between pixels in an image images. For each pixel,
outrow[j][i] = inrow[j][i+1] - inrow[j][i];
is computed.

fldx_forw.c

Forward difference

int fldx_forw(img1,img2)
    IFSIMG img1,img2;

Compute the forward difference between pixels in an image images. For each pixel,
  outrow[j][i] = inrow[j][i] - inrow[j][i-1];

fldxx.c

Second partial derivative with respect to x.

int fldxx(img1,img2)
    IFSIMG img1,img2;

Uses a 3x1 kernel.

fldxy.c

Second partial derivative with respect to x and y.

int fldxy(img1,img2)
    IFSIMG img1,img2;

Uses a 3x3 neighborhood.

fldxz.c

Second partial wrt x,z

int fldxz(img1,img2)
    IFSIMG img1,img2;

fldy.c

First partial derivative with respect to y.
int fldy(img1,img2)
    IFSIMG img1,img2;

fldy_back.c

Backward difference
The forward and backward differences both estimate the first partial derivative with respect to y, but do it with higher resolution (and higher noise sensitivity) than fldy. They are useful in iterative algorithms where you do one iteration using the forward difference, and the next using the backward.
int fldy_back(img1,img2)
    IFSIMG img1,img2;
Compute the backward difference between pixels in an image images. For each pixel,
 outrow[j][i] = (inrow[j+1][i] - inrow[j-1][i])*.5;
is computed.

fldy_forw.c

Forward difference
int fldy_forw(img1,img2)
    IFSIMG img1,img2;
Compute the forward difference between pixels in an image images. For each pixel,
outrow[j][i] = inrow[j][i] - inrow[j-1][i];

fldyy.c

Second partial derivative with respect to y.
int fldyy(img1,img2)
    IFSIMG img1,img2;
Uses a 3x1 kernel.

fldyz.c

Second partial derivative with respect to y and z.
int fldyz(img1,img2)
    IFSIMG img1,img2;
Uses a 3x3 kernel. Obviously, only meaningful for 3D data

fldz.c

First partial derivative with respect to z. Obviously, only meaningful for 3D data.
int fldz(img1,img2)
    IFSIMG img1,img2;

fldzz.c

Second partial derivative with respect to z. Only meaningful for 3D data.
int fldzz(img1,img2)
    IFSIMG img1,img2;
Uses a 3x1 kernel.

flexp.c

Exponentiate
int flexp(img1,img2)
    IFSIMG img1,img2;
Each pixel of img2 is independently exponentiated. Result stored in img2.

flgrow.c

flln.c

Compute logs of pixels independently
int flln(img1,img2)
    IFSIMG img1,img2;

Computes out[i] = log(in[i]);

flmults.c

Scalar multiply
int flmults(img1,img2,scalar)
    IFSIMG img1,img2;
    float scalar;
Multiply each pixel of img1 by scalar and store the result in the corresponding pixel of img2.

flmultv.c

Vector multiply
int flmultv(img1,img2,img3)
    IFSIMG img1,img2,img3;
Each pixel of img1 is multiplied by the corresponding pixel of img2, the result stored in img3.

flneg.c

Negate image pixels
int flneg(img1,img2)
    IFSIMG img1,img2;
Each pixel of img1 is multiplied by -1 and the result stored in img2.

flnorm.c

int flnorm(img1,norm)
    IFSIMG img1;
    float *norm;
Returns the 2-norm of the image img1. The second argument is a POINTER to a float, and the norm (square root of sum of squares of pixels) will be returned.

flone_border.c

int flone_border(img1,img2)
    IFSIMG img1,img2;

Sets the left-most and right-most columns of the image to one. Then set the top row and bottom row to one. Result copied into img2.

flpad.c

Image pad
int flpad(img1,img2)
    IFSIMG img1,img2;
Input image img1 is padded by taking the leftmost pixel on each row and replacing it by the next-to-leftmost. Same thing on right side, top, and bottom. If 3D, appropriate things are done to the first and last frame.

flplanar.c

Image pad using linear interpolation.
int flplanar(img1,img2)
    IFSIMG img1,img2;
The leftmost pixel on each line is replaced by what it would be if the first three pixels were linearly increasing in brightness -- the zeroth pixel is the linear interpolation of the first and second. Similar on right side, top, and bottom. Only implemented for 2D images.

flrec.c

Reciprocal of an image
int flrec(img1,img2)
    IFSIMG img1,img2;
Each pixel of the input image, img1, is divided into one, and the result stored in the corresponding pixel of img2. This function tests for divide by zero, reports an error, and exits early.

flshx.c

Shift
int flshx(img1,dx,img2)
    int   dx;
    IFSIMG img1,img2;
input image img1 is shifted by dx pixels in the x direction and the result stored in img2. Operation is valid for 3D images.

flshxy.c

Shift image
int flshxy(img1,dx,dy,img2)
    int   dx,dy;
    IFSIMG img1,img2;

Input image img1 is shifted dx pixels in x and dy pixels in y. Operation is valid for 3D images.

flshy.c

Shift image
int flshy(img1,dy,img2)
    int   dy;
    IFSIMG img1,img2;
Shifts image in y direction. Operation is valid for 3D images.

flsq.c

Square pixel values
int flsq(img1,img2)
    IFSIMG img1,img2;
For every pixel, out[i] = in[i] * in[i] ;

flsqrt.c

Square root of pixels
int flsqrt(img1,img2)
    IFSIMG img1,img2;
Each pixel in the output image img2 is the square root of the corresponding pixel in the input image img1. The input image is tested for negative values. A negative value will the reported and the subroutine will terminate early.

flsubs.c

Scalar subtract
int flsubs(img1,img2,scalar)
    IFSIMG img1,img2;
    float scalar;

The scalar value scalar is subtracted from each pixel in the input image, img1. Result is stored in the corresponding pixel of the output image img2.

flsubv.c

Vector subtract
Each pixel in the second image, img2 is subtracted from the corresponding pixel in img1, and the result stored in img3. Valid for any number of dimensions.

flthresh.c

Threshold
int flthresh(img1,img2,scalar,bkgnd)
    IFSIMG img1,img2;
    float scalar,bkgnd;
Each pixel of input image img1 is tested against scalar. If pixel > scalar, the pixel is unchanged, otherwise, the corresponding pixel in the output image is set to the value bkgnd. Valid for any number of dimensions.

flzero_border.c

Set border pixels to zero.
int flzero_border(img1,img2)
    IFSIMG img1,img2;
Valid for two dimensional images.