EIKONA

Digital image processing package

Thessaloniki 1997

Manual, Part IV

Modules

The JPEG input/output routines were based in part on the work of the Independent JPEG Group.

The GIF, BMP, TGA input/output routines were base in part on the PBMPLUS toolkit.

The scanner interface was based in part on the TWAIN Toolkit, Release 1.6.
The TWAIN toolkit is distributed as is. The developer and the distributors of the TWAIN toolkit expressly disclaim all implied, express or statutory warranties including, without limitation, the implied warranties of merchantability, noninfringement of third party rights and fitness for a particular purpose. Neither the developers nor the distributors will be liable for damages, whether direct, indirect, special, incidental, or consequential as a result of the reproduction, modification, distribution or other use of the TWAIN toolkit.

Distributor :

Contents

1

EIKONA for Arts Module

1.1 Introduction

EIKONA for Arts is a powerful extension of EIKONA. It contains a mosaicing algorithm for automatic, semi-automatic or manual image mosaicing, and two image registration algorithms. Section 1.3 describes the library functions for the arts module. Section 1.2 is a brief user's guide of the Arts module for EIKONA.

1.2 User's guide

1.2.1 Installation

Installation of the "EIKONA for Arts" module is particularly simple. Just copy the "`arts.dll`" file in the directory where EIKONA is installed and run EIKONA. The module will be loaded automatically by EIKONA.

1.2.2 Automatic mosaicing

Automatic image mosaicing is performed on a set of equal size images having overlapping parts in the horizontal and/or in the vertical direction, and creates an output image as the result of the mosaicing of the input images. The images must have the same root filename, and must be characterized by the number of

the relevant scanning order. The user must provide the root filename of the source images, along with information concerning the overlapping regions. In the following we suppose that we have a set of six images, namely x1.jpg, x2.jpg, ...,x6.jpg, having certain overlapping parts. These images are supposed to form a 2×3 grid. Figure 1.2.1 shows image "x1.jpg" from this set.

Figure 1.2.1 Mosaic source image "x1.jpg".

Automatic image mosaicing can be performed by selecting the option **Modules, Arts, Automatic Mosaic** which brings up the *Automatic Mosaic* dialog box depicted in figure 1.2.2 . The upper half of the dialog is used just as a common

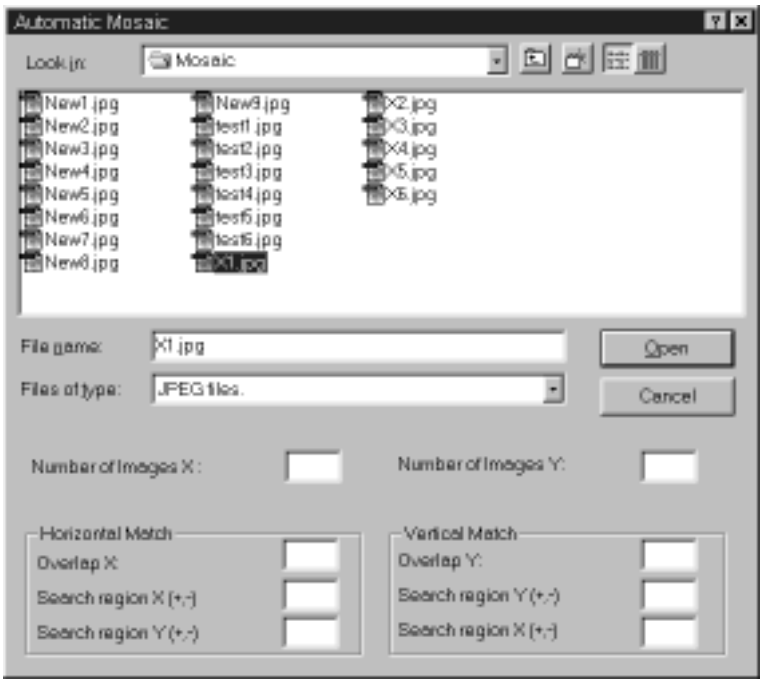

Figure 1.2.2 The Aytomatic Mosaic dialog box.

Open dialog box. We use it to select one of the images of the set in the *File Name* field (e.g. 'x2.jpg'). In the *Number of Images* fields we specify the horizontal and vertical dimensions of the grid of images to be mosaiced 2×3 for our example set. In the *Horizontal Match group*, we provide the amount of overlap in the X dimension, for example 50, and the size of the search regions in the X and Y dimensions, 10 and 10, respectively. This means that the search is performed in the range [45, 55] in the horizontal direction and [-5, 5] in the vertical direction. Accordingly, in the *Vertical Match* group we provide the amount of overlap in the Y dimension, also 50, and the size of the search regions in the X and Y dimensions, again 10 and 10, respectively. Press the *Mosaic* button mosaic the images. The result is shown in Figure 1.2.3.

Figure 1.2.3 Automatic mosaic output of image set "x".

1.2.3 Manual image mosaicing

Manual image mosaic can be performed by using the option **Modules, Arts, Manual Mosaic**. This brings up the *Manual Mosaic* dialog box depicted in Figure 1.2.4. The dialog box resembles the one for automatic mosaicing the only difference being

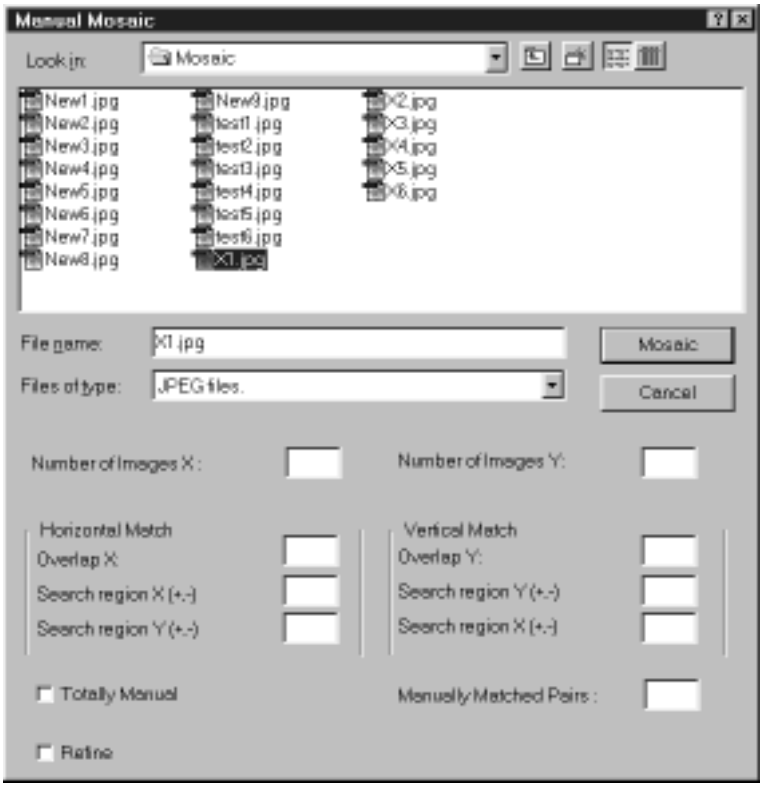

Figure 1.2.4 The Manual Mosaic dialog box.

the controls at the bottom of the dialog box. In manual mosaic, the user can specify manually the corresponding points in one or more pairs of overlapping images. The number of pairs to be matched manually is specified in the *Manually matched pairs* edit box. The rest of the image pairs are matched automatically. The user can choose to match all image pairs manually, by checking the *Totally Manual* toggle button . In this case the user must specify matching points for all pairs of overlapping images. Finally the *Refine* button specifies whether a local refinement of the point coordinates will be performed.

Continuing our example with the "x1.jpg"... "x6.jpg" set we may set the same parameters for the horizontal match and vertical match as in the automatic mosaic case and enter 2 in the *Manually matched pairs* field. That means that we must specify *exactly* four matching points in two overlapping image pairs. We check the *Refine* toggle and we leave the *Totally Manual* toggle unset.

After pressing the *Mosaic* button, a window which contains a grid of all the

source images is displayed along with the *Select Pixels* dialog box. The image grid window is shown in Figure 1.2.5. From the image grid, we select any two pairs of

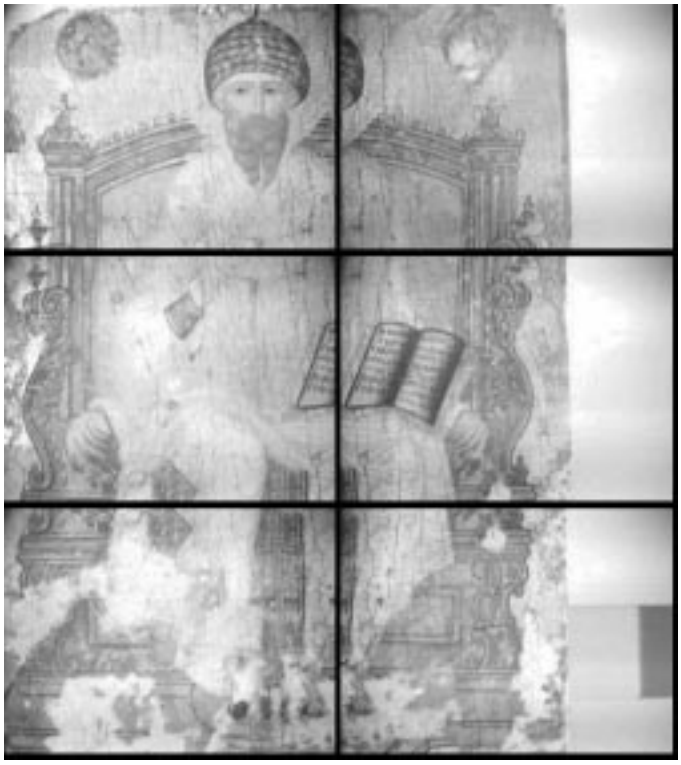

Figure 1.2.5 Grid of source images in manual mosaic.

images which we want to match manually. Then we select the matching points by clicking the left mouse button on the same point in both images. The *Select Pixels* dialog shows a magnified view of the region under the cursor when the cursor is over the image grid window. Valid points are signified by a green cross appearing in the magnified view ; a large red 'X' sign means that the point under the cursor is not valid. If we specify more than four points, a warning beep will be heard and the coordinates of the last point will be replaced by the coordinates of the new point specified.

After specifying all the points, we press the *OK* button in the *Select Pixels* dialog box. The result of the manual mosaic routine is shown in Figure 1.2.6.

Figure 1.2.6 Manual mosaic output of image set "x".

1.2.4 Simple Image Registration

Registration is performed on two source images , with the first image considered as the *reference* image.Registration produces an output image which is a scaled, translated and rotated version of the second source image, based on certain feature points specified by the user.

Let us suppose that we have two source images, namely "Visible" and "X-ray", shown in Figure 1.2.1.

Figure 1.2.1 Reference source image "Visible" and source image "X-ray" used for image registration.

Image registration is performed by using the option **Modules, Arts, Simple Registration** which brings up the *Image Registration* dialog box shown in Figure 1.2.2.

Figure 1.2.2 The Simple Registration dialog box.

We specify the two source images in the *Source Image* fields. The *Source Image 1* specifies the reference image while the *Source Image 2* field specifies the image on which scaling, rotation and translation transformations will be applied in order for the result to match the reference buffer.In the *Destination Image* field we specify the output image (we can ask for the creation of a new image through the <New Buffer> entry). We must specify a displacement window size (13×13 is usually a good value), and a comparison window size which should be considerably larger (in the case mentioned, 19×19). In the *Feature Points* field we enter the number of feature points we will use. We may specify the threshold used for the criterion of retaining a certain feature point through the *Rejection Factor* field. A value of 0 would indicate that all the feature points given would be kept, while a value of 1 would indicate that all would be rejected (a good value would be 0.1 – 0.2). If we want to automatically improve the quality of the registration using local matching from the image itself, we must set the *Refine Matching* button. Finally, in the *Scaling options* group we may specify the type of scaling the algorithm should use. Choose *Independent Scaling* to allow independent scaling factors for

the horizontal and vertical image dimensions, or *Uniform scaling* if the same scaling factor should be used for both image dimensions. After pressing the *OK* button a modeless dialog box appears. We can now specify the feature points by clicking on the two source images with the left mouse button (the keyboard is also available for fine movements ; use the arrow keys to move and the space bar to select). Feature points should be specified either in pairs (corresponding points on the first and second source images) or in order (all feature points in one image then all feature points in the other image in the same order). After clicking the *OK* button the result should resemble Figure 1.2.3.

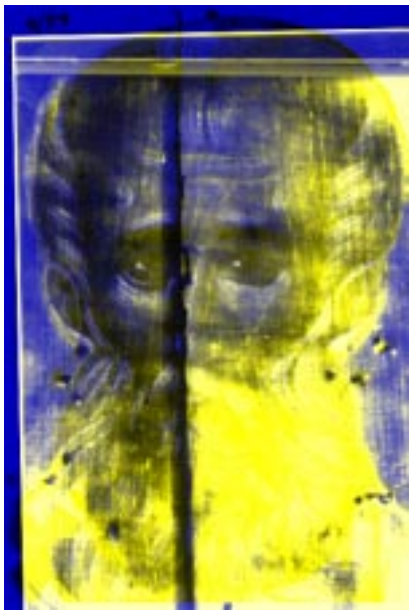

Figure 1.2.3 Registration output image.

1.2.5 Combined Registration

Combined Registration is an advanced version of registration. It differs from the basic version of registration in a number of ways. The most important is that the result is a combination of the two source images (hence the name), which can be computed in two ways either averaging the two images where they overlap, and keeping their values where they don't (merging), or subtracting the images and adding a bias (in order to preserve the negative values) where the images overlap and keeping their values where they don't (subtracting). A second difference is the use of mask images to define arbitrarily shaped regions of interest. These masks are white where there is image information, black where there is none and have intermediate values in border areas. Generally, for most images we only need to

define an all-white mask. The third difference is that that the refined feature point rejection procedure has been improved. All the user has to do is to select the number of feature points he wants to keep. The best feature points up to this number will be kept, and the rest will be rejected. A minor fourth difference is that the user now has the ability to disable scaling and keep the size of the image the same.

Combined registration can be performed by selecting **Modules, Arts, Combined Registration**. In the dialog box that appears we specify the source buffers along with masks that define arbitrarily shaped regions of interest on them. In the *Displacement Window* and *Comparison Window* groups we enter the dimensions of the displacement and comparison window respectively. We enter the number of feature points in the *Feature Points* field and the number of them to keep in the *Feature Points to Keep* field. From the *Combine Options* group we select whether the images should be merged or subtracted as described above. Scaling can be toggled on or off with the *Scaling* button. If it is checked we should specify the type of scaling from the *Scaling Options* group. Finally, the *Refine Matching* button toggles the local refinement step on or off. After filling these fields and pressing the *OK* button the specified number of feature points should be marked on the two images as described in the **Simple Registration** case. The output consists of the combined image and a corresponding mask to facilitate further registration steps.

As an example of combined registration consider the two microscopy images shown in Figure 1.2.4. Creating an all-white mask for them, the combined regitration output with independent scaling turned on and refinement enabled is shown in Figure 1.2.5 along with the output mask.

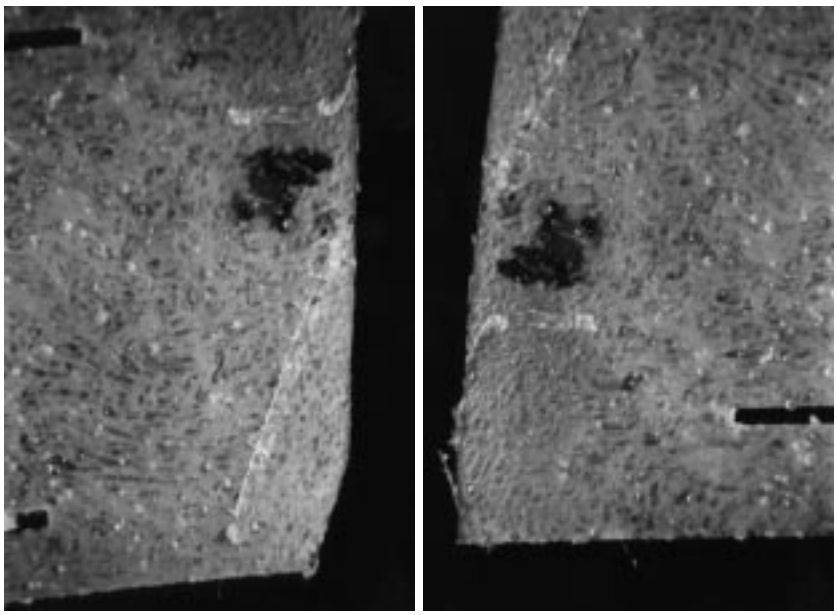

Figure 1.2.4 Source images for combined registration.

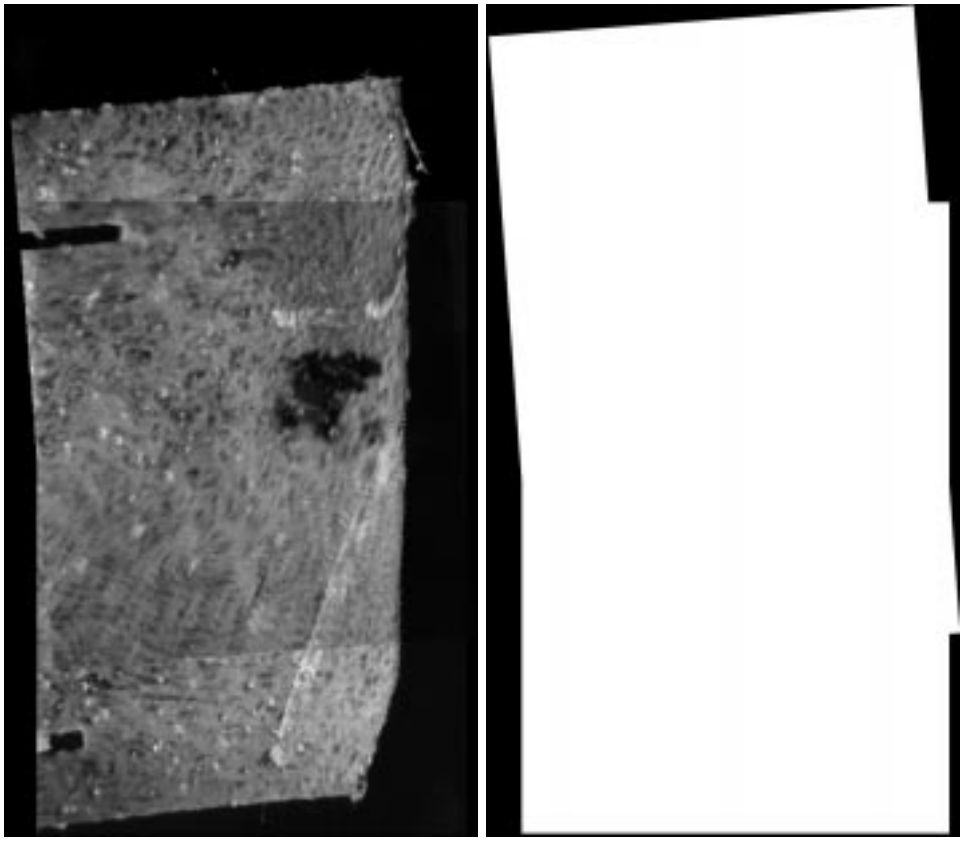

Figure 1.2.5 Combined registration output image and mask.

1.3 Library functions

1.3.1 adjust

Subroutine to equalize luminance and contrast of two images.
LIBRARY MODE:

int adjust (img1, mask1, img2, mask2, ymax1, xmax1, ymax2, xmax2, dx, dy)

image img1, mask1, img2, mask2;
int ymax1, xmax1, ymax2, xmax2, dx, dy;

img1	: reference image
mask1	: mask containing arbitrarily shaped ROI in reference image
img2	: second image
mask2	: mask containing arbitrarily shaped ROI in second image
ymax1, xmax1	: size of reference image
ymax2, xmax2	: size of second image
dx,dy	: base coordinate difference between the two images

SHORT DESCRIPTION: This routine normalizes the gray values of image img2 so that their mean and standard deviation is equal to those of a reference image img1. The above are computed using only those pixels whose mask i1_mask,i2_mask is set.
RETURN VALUES: 0, -90
SEE ALSO: comb_registr, comb_registr_is

1.3.2 Check_misimage

Subroutine to check the validity of values of misimage.
LIBRARY MODE:

int Check_misimage (misimage, No_pairs, No_Im_X, No_Im_Y)

FINDIMAG *misimage;
int No_pairs;

int No_Im_X,No_Im_Y;

misimage : Pointer to misimage structure.
No_pairs : Number of pairs of images for manually mosaic.
No_Im_X, No_Im_Y : Number of images on X and Y axes.

SHORT DESCRIPTION: This function is used for checking the validity of values of misimage. Structure misimage contains the corresponding points in the image pairs to be matched manually in manual image mosaic. If the user gives two images that cannot combined, this function will return an error. It should be called just after the Re_Order_misimage function.
RETURN VALUES: 0, -3, -12, -90
SEE ALSO: Mosaic, Re_Order_misimage, refine_MM

1.3.3 image_merge

Routine to merge two images of different shapes.
LIBRARY MODE:

void image_merge (img1, mask1, ymax1,xmax1,
img2,mask2, ymax2, xmax2, dx,dy)

image img1, mask1;
int ymax1, xmax1;
image img2, mask2;
int ymax2, xmax2, dx, dy;

img1 : image to be transferred
mask1 : corresponding mask
ymax1, xmax1 : corresponding size
img2 : destination image
mask2 : corresponding mask
ymax2, xmax2 : corresponding size
dx, dy : base coordinate difference between the two images

SHORT DESCRIPTION: This routine merges two images img1,img2 whose regions of interest are defined by masks mask1,mask2 and stores the result in img2. If both images contain information at a pixel, the resulting image pixel is their average. If only one image contains information at a pixel, it remains

unchanged in the result image. The resulting image mask is the union of the original masks.
SEE ALSO: comb_registr, comb_registr_is

1.3.4 *image_subtract*

Routine to subtract two images of different shapes.
LIBRARY MODE:

```
void image_subtract ( img1, mask1, ymax1, xmax1,
                      img2, mask2, ymax2, xmax2, dx, dy)

image img1, mask1;
int   ymax1, xmax1;
image img2, mask2;
int   ymax2, xmax2, dx, dy;
```

img1	: image to be transferred
mask1	: corresponding mask
ymax1, xmax1	: corresponding size
img2	: destination image
mask2	: corresponding mask
ymax2, xmax2	: corresponding size
dx, dy	: base coordinate difference between the two images

SHORT DESCRIPTION: This routine subtracts two images img1,img2 whose regions of interest are defined by masks mask1,mask2 and stores the result in img2. If both images contain information at a pixel, the resulting image pixel is their difference, with 0 being represented by gray and negative being darker. If only one image contains information at a pixel, it remains unchanged in the result image. The resulting image mask is the union of the original masks.
SEE ALSO: comb_registr, comb_registr_is

1.3.5 *Mosaic()*

Subroutine to perform mosaicing of a set of images.

LIBRARY MODE:

```
int Mosaic ( K1, K2, tim, Sear_X, Sear_Y, Var_X, Var_Y,
             Sear2_X, Sear2_Y, ManImages, misimage,
             roitop, roileft, roibottom, roiright,
             Min_X, Max_X, Min_Y, Max_Y, WhereIsIt )

int             K1, K2;
image           *tim;
int             Sear_X, Sear_Y, Var_X, Var_Y, Sear2_X , Sear2_Y;
int             ManImages;
FINDIMAG        *misimage;
int             roitop, roileft, roibottom, roiright;
int             *Min_X, *Max_X, *Min_Y, *Max_Y;
unit            **WhereIsIt;
```

K1	: number of images on Y coordinate.
K2	: number of images on X coordinate.
tim	: pointer to the images to be mosaiced.
Sear_X	: search region on X axis around the X overlapped area for matching the images horizontally.
Sear_Y	: search region on Y axis around the Y overlapped area for matching the images vertically.
Var_X	: The overlapped area on X axis
Var_Y	: The overlapped area on Y axis
Sear2_X	: Search region on X axis for matching the images vertically
Sear2_Y	: Search region on Y axis for matching the images horizontally.
ManImages	: Number of pairs of images for manual mosaic.
misimage	: pointer to the structure with the images to be handled manually.
roitop,roileft, roibottom,roiright	: region of interest
Min_X	: the leftmost coordinate of the image
Max_X	: the rightmost coordinate of the image
Min_Y	: the top coordinate of the image
Max_Y	: the bottom coordinate of the image
WhereIsIt	: the transpositions of the origin point of each image with respect to the origin of the final mosaicked image

SHORT DESCRIPTION: This subroutine computes a matrix of transpositions by applying mosaicing on a set of component images. This matrix is composed of

the transpositions of the origin of each component image from the origin of the final mosaicked image. The component images should have the same size. They should also have overlapping parts on the horizontal and on the vertical direction. The naming convention for the component images is *fnamei.jpg* where *i* is the image index in the scanning order. To give an example, the names of 6 component images that form two rows of three images when the root name is "image" should be "image1.jpg", "image2.jpg", "image3.jpg" (for the images in the first row) "image4.jpg", "image5.jpg", "image6.jpg" (for the images in the second row). The output image will be named "image.jpg". Name creation, opening and reading the component images as well as saving the output image is done externally of the routine. The output image will be created from the WriteTheImage function just after calling the Mosaic routine. The mosaicing is performed by finding the position where the best match between two adjacent images occurs. The regions where the search for the best match is performed are controlled by the variables Pixels in overlapping regions are weighted in order to smooth the transitions between regions with different graylevels. The mosaicing can be performed on color or BW images. The quality of the output JPEG image is controlled by the integer variable "quality". Mosaicing of color images requires two more image buffers for temporary storage. The size of these buffers should be equal with the size of the component images. In the case of "Manually Mosaic" the user will be prompt to specify the points (and consequently the corresponding images) to be combined manually. If the user gives wrong points then the process will be stopped. You must not give images that can not be combined and also you must give points for all of pairs of images that you had specified before. If you select refine then the refinement function will try to find the precise corresponding points in every pair of images.
RETURN VALUE: 0, -1, -29, -90, -300
SEE ALSO: fromjpeg, tojpeg

1.3.6 OverlapWest

Subroutine to find the matching between two images.
LIBRARY MODE:

```
int OverlapWest ( im_r, im_c, POZ_X, POZ_Y, M, N, Sear_X, Sear_Y,
                  Var_X, Var_Y, Sear2_X, Sear2_Y, mean1, mean2,
                  MIN, Ave, WhereIsIt )

image   im_r, im_c;
int     POZ_X, POZ_Y;
```

```
int     M, N;
int     Sear_X, Sear_Y, Var_X, Var_Y, Sear2_X, Sear2_Y;
int     mean1, mean2;
float   *MIN;
float   *Avel;
unit    ** WhereIsIt;
```

im_r	: left image to be matched
im_c	: right image to be matched
POZ_X	: the image rank on X axis
POZ_Y	: the image rank on Y axis
M	: the X size of an initial image
N	: the Y size of an initial image
Sear_X	: search region on X axis around the X overlapped area for matching the images horizontally.
Sear_Y	: search region on Y axis around the Y overlapped area for matching the images vertically.
Var_X	: The overlapped area on X axis
Var_Y	: The overlapped area on Y axis
Sear2_X	: search region on X axis for matching the images vertically.
Sear2_Y	: search region on Y axis for matching the images horizontally.
mean1	: the mean of the left image
mean2	: the mean of the right image
MIN	: the difference on the common part between the images
Ave	: evaluation of the variance of the common region calculated as a confidence measure in the result
WhereIsIt	: the transpositions of the origin point of each image with respect to the origin of the final mosaicked image.

SHORT DESCRIPTION: This subroutine finds the matching between two images which are supposed to have a common band, one at the right side and the other at the left side.
RETURN VALUES: 0
SEE ALSO: Mosaic, OverlapNorth

1.3.7 OverlapNorth

Subroutine to find the matching between two images.

LIBRARY MODE:

```
int OverlapNorth ( im_r, im_c, POZ_X, POZ_Y, M, N, Sear_X, Sear_Y,
                   Var_X, Var_Y, Sear2_X, Sear2_Y, mean1, mean2,
                   MIN, Ave, WhereIsIt )

image    im_r, im_c;
int      POZ_X, POZ_Y;
int      M, int N;
int      Sear_X, Sear_Y, Var_X, Var_Y, Sear2_X, Sear2_Y;
int      mean1, mean2;
float    *MIN;
float    *Avel;
unit     **WhereIsIt;
```

im_r	:	down image to be matched
im_c	:	upper image to be matched
POZ_X	:	the image rank on X axis
POZ_Y	:	the image rank on Y axis
M	:	the X size of an initial image
N	:	the Y size of an initial image
Sear_X	:	search region on X axis around the X overlapped area for matching the images horizontally.
Sear_Y	:	search region on Y axis around the Y overlapped area for matching the images vertically.
Var_X	:	The overlapped area on X axis
Var_Y	:	The overlapped area on Y axis
Sear2_X	:	search region on X axis for matching the images vertically
Sear2_Y	:	search region on Y axis for matching the images horizontally.
mean1	:	the mean of the left image
mean2	:	the mean of the right image
MIN	:	the difference on the common part between the images
Ave	:	evaluation of the variance of the common region calculated as a confidence measure in the result
WhereIsIt	:	the transpositions of the origin point of each image with respect to the origin of the final mosaicked image.

SHORT DESCRIPTION: This subroutine finds the matching between two images which are supposed to have a common band, one at the bottom side and the other at the top side.

RETURN VALUES: 0

SEE ALSO: Mosaic, OverlapWest

1.3.8 Re_Order_misimage

Subroutine to sort the elements of the misimage. The misimage points to an array of structures each of them consists of the numbers of images to be mosaicked with the corresponding common points of the two images.
LIBRARY MODE:

int Re_Order_misimage (misimage, No_pairs)

```
FINDIMAG   *misimage;
int        No_pairs;
```

misimage : Pointer to misimage structure.
No_pairs : Number of pairs of images for manually mosaic.

SHORT DESCRIPTION: This function reorders the pairs of images of each structure of misimage array in descending order.
RETURN VALUES: 0, -3, -12
SEE ALSO: Mosaic, Check_misimage, refine_MM

1.3.9 refine_MM

Subroutine to refine the points that the user gives on each pair of images
LIBRARY MODE:

```
int refine_MM ( tim, NoImages, misimage,
                roitop, roileft, roibottom, roiright,
                Block, Search )
```

```
image      *tim;
int        NoImages;
FINDIMAG   *misimage;
int        roitop, roileft, roibottom, roiright;
int        Block, Search;
```

tim : pointer to the images to be mosaicked.

NoImages	: Number of pairs of images for manual mosaic.
misimage	: pointer to the structure with the images to be handled manually.
roitop, roileft, roibottom, roiright	: region of interest
Block, Search	: The dimensions of the block and the search area for the block match algorithm.

SHORT DESCRIPTION: This function uses the block match algorithm to refine the points that the user gives with the mouse, because it is almost impossible for the user to select the exact corresponding points.
RETURN VALUES: 0, -1
SEE ALSO: Mosaic, Re_Order_misimage, Check_misimage

1.3.10 registr()

Subroutine to perform registration of two images given a set of user defined feature points. This function uses the same scaling factor for the x and y dimensions of an image.
LIBRARY MODE:

```
int registr ( feat1x, feat1y, feat2x, feat2y, fs, wcwx, wcwy,
              wdwx, wdwy, img1, i1_y1, i1_x1, i1_y2, i1_x2, img2,
              i2_y1, i2_x1, i2_y2, i2_x2, reslt, type, qual )

double   *feat1x, *feat1y, *feat2x, *feat2y;
int      fs, wcwx, wcwy, wdwx, wdwy;
image    img1;
int      i1_y1, i1_x1, i1_y2, i1_x2;
image    img2;
int      i2_y1, i2_x1, i2_y2, i2_x2;
image    reslt;
int      type;
double   qual;
```

feat1x, feat1y	: feature point coordinates arrays (image 1)
feat2x, feat2y	: feature point coordinates arrays (image 2)
fs	: number of feature point pairs
wcwx, wcwy	: comparison window size (odd)

wdwx, wdwy : displacement window size (odd)
img1 : input image 1 (reference image)
i1_y1, i1_x1 : upper left corner coordinates (image 1)
i1_y2, i1_x2 : lower right corner coordinates (image 1)
img2 : input image 2
i2_y1, i2_x1 : upper left corner coordinates (image 2)
i2_y2, i2_x2 : lower right corner coordinates (image 2)
reslt : output image
type : registration indicator (0=use only given feature points, 1=refine matching)
qual : feature point rejection factor

SHORT DESCRIPTION: This subroutine performs registration of two BW images based on a set of feature points. Image img2 is scaled, translated and rotated so as to match image img1 (reference image). The result is stored in image img3. The coordinates of the feature points should be provided by the user in two one-dimensional matrices i.e. feat1x, feat1y for the feature points in image img1 and feat2x, feat2y for the feature points in image img2. The coordinates of the i-th pair of feature points should be stored on the i-th matrix element in all four matrices. The registration is based merely on the user-provided feature points (type=0). Alternatively, a second refining step where a window (comparison window) centered on each feature point is used to search for a better match in a neighborhood (displacement window) around the corresponding point in the other image, can be applied (type=1). Both the comparison and the displacement window should be of odd size. Pairs of feature points that do not exhibit good match are automatically rejected by the algorithm. The rejection ratio is controlled by parameter qual which ranges from 0 (keep all pairs of feature points) to 1 (reject all pairs of feature points).
RETURN VALUE: 0, -1, -9, -24, -40
SEE ALSO: registr_is, comb_registr, comb_registr_is

1.3.11 registr_is()

Subroutine to perform registration of two images given a set of user defined feature points. This function uses independent scaling (different scaling factors for the x and y dimensions of an image).
LIBRARY MODE:

int registr_is (feat1x, feat1y, feat2x, feat2y, fs, wcwx, wcwy,

```
                wdwx, wdwy, img1, i1_y1, i1_x1, i1_y2, i1_x2, img2,
                i2_y1, i2_x1, i2_y2, i2_x2, reslt, type, qual )

double   *feat1x, *feat1y, *feat2x, *feat2y;
int      fs, wcwx, wcwy, wdwx, wdwy;
image    img1;
int      i1_y1, i1_x1, i1_y2, i1_x2;
image    img2;
int      i2_y1, i2_x1, i2_y2, i2_x2;
image    reslt;
int      type;
double   qual;

feat1x, feat1y   : feature point coordinates arrays (image 1)
feat2x, feat2y   : feature point coordinates arrays (image 2)
fs               : number of feature point pairs
wcwx, wcwy       : comparison window size (odd)
wdwx, wdwy       : displacement window size (odd)
img1             : input image 1 (reference image)
i1_y1, i1_x1     : upper left corner coordinates (image 1)
i1_y2, i1_x2     : lower right corner coordinates (image 1)
img2             : input image 2
i2_y1, i2_x1     : upper left corner coordinates (image 2)
i2_y2, i2_x2     : lower right corner coordinates (image 2)
reslt            : output image
type             : registration indicator ( 0=use only given feature points,
                                            1=refine matching)
qual             : feature point rejection factor
```

SHORT DESCRIPTION: This subroutine performs registration of two BW images based on a set of feature points. Image img2 is scaled, translated and rotated so as to match image img1 (reference image). The result is stored in image img3. The coordinates of the feature points should be provided by the user in two one-dimensional matrices i.e. feat1x, feat1y for the feature points in image img1 and feat2x, feat2y for the feature points in image img2. The coordinates of the i-th pair of feature points should be stored on the i-th matrix element in all four matrices. The registration is based merely on the user-provided feature points (type=0). Alternatively, a second refining step where a window (comparison window) centered on each feature point is used to search for a better match in a neighborhood (displacement window) around the corresponding point in the other image, can be applied (type=1). Both the comparison and the displacement window should be of odd size. Pairs of feature points that do not exhibit good match are automatically rejected by the algorithm. The rejection

ratio is controlled by parameter qual which ranges from 0 (keep all pairs of feature points) to 1 (reject all pairs of feature points).
RETURN VALUE: 0, -1, -9, -24, -40
SEE ALSO: registr, comb_registr, comb_registr_is

1.3.12 comb_registr()

Routine to register one image to another, allocate space for their combined representation, and create appropriate masks. The function uses the same scaling factor for both image dimensions.
LIBRARY MODE:

```
int comb_registr ( feat1x, feat1y, feat2x, feat2y, fs, wcwx, wcwy,
                   wdwx, wdwy, img1, i1_mask, i1_ymax, i1_xmax,
                   img2, i2_mask, i2_ymax, i2_xmax, reslt, res_mask,
                   res_ymax, res_xmax, type, qual, dx, dy, scale )

double   *feat1x, *feat1y, *feat2x, *feat2y;
int      fs, wcwx, wcwy, wdwx, wdwy;
image    img1, i1_mask;
int      i1_ymax, i1_xmax;
image    img2, i2_mask;
int      i2_ymax, i2_xmax;
image    *reslt, *res_mask;
int      *res_ymax, *res_xmax;
int      type, qual, *dx, *dy, scale;
```

feat1x, feat1y	: arrays containing feature points in the reference image
feat2x, feat2y	: arrays containing feature points in the second image
fs	: number of feature points
wcwx, wcwy	: size of comparison window
wdwx, wdwy	: size of displacement window
img1	: reference image
i1_mask	: mask containing arbitrarily shaped ROI in reference image
i1_ymax, i1_xmax	: size of reference image
img2	: second image
i2_mask	: mask containing arbitrarily shaped ROI in second image
i2_ymax, i2_xmax	: size of second image

reslt	: result image
res_mask	: mask containing arbitrarily shaped ROI in result image
res_ymax, res_xmax	: size of result image
type	: 0–unrefined registration/ 1–refined registration
qual	: number of feature points to keep
dx, dy	: base coordinate difference between the two images
scale	: 0–do not use scaling/1–use scaling

SHORT DESCRIPTION: The object of the routine is to rotate, translate and scale an image img2 so as to coincide with another (reference) image img1. The correspondence is determined by two sets of feature points feat1x, feat1y, feat2x, feat2y. The computation is performed in two steps. First, the scaling, translation and rotation are computed from the given feature points. Then, the affine transform of img2 is performed according to the computed parameters. The transformed image is stored in a buffer big enough to ensure that no information is lost. If the flag type is 0, the second step is not performed and the execution of the routine stops here. If the flag is 1, the routine proceeds to the refinement of the feature points. The refinement is performed by finding the optimal displacement for each pair of feature points so that the difference of the windows surrounding them is minimized. The difference is computed as the RMS of the difference of the values of the corresponding pixels in the windows, normalized according to mean and standard deviation in each window. The size of the window within which the comparison happens is given by wcwx,wcwy and the range of the displacement tried for each feature point pair by wdwx,wdwy. After the refinement, the feature point sets are sorted according to the similarity of their surrounding regions and only the `qual` best are used for the following. Then, the scaling, translation and rotation are computed from the refined feature points, and img2 is transformed according to them. The transformed image is stored in a buffer big enough to ensure that no information is lost, and is returned to the calling routine. In all the above, only the pixels for which the corresponding mask i1_mask,i2_mask is set are processed.
RETURN VALUE: 0, -1, -9, -24, -40
SEE ALSO: registr, registr_is, comb_registr_is.

1.3.13 comb_registr_is()

Routine to register one image to another, allocate space for their combined representation, and create appropriate masks. The function uses independent scaling factors for the two image dimensions.

LIBRARY MODE:

```
int comb_registr_is ( feat1x, feat1y, feat2x, feat2y, fs, wcwx, wcwy,
                      wdwx, wdwy, img1,i1_mask, i1_ymax, i1_xmax,
                      img2, i2_mask,i2_ymax, i2_xmax, reslt, res_mask,
                      res_ymax, res_xmax, type, qual, dx, dy, scale )

double   *feat1x, *feat1y, *feat2x, *feat2y;
int      fs, wcwx, wcwy, wdwx, wdwy;
image    img1, i1_mask;
int      i1_ymax, i1_xmax;
image    img2, i2_mask;
int      i2_ymax, i2_xmax;
image    *reslt, *res_mask;
int      *res_ymax, *res_xmax;
int      type, qual, *dx, *dy, scale;
```

feat1x, feat1y	: arrays containing feature points in the reference image
feat2x,feat2y	: arrays containing feature points in the second image
fs	: number of feature points
wcwx, wcwy	: size of comparison window
wdwx, wdwy	: size of displacement window
img1	: reference image
i1_mask	: mask containing arbitrarily shaped ROI in reference image
i1_ymax, i1_xmax	: size of reference image
img2	: second image
i2_mask	: mask containing arbitrarily shaped ROI in second image
i2_ymax, i2_xmax	: size of second image
reslt	: result image
res_mask	: mask containing arbitrarily shaped ROI in result image
res_ymax, res_xmax	: size of result image
type	: 0–unrefined registration/ 1–refined registration
qual	: number of feature points to keep
dx, dy	: base coordinate difference between the two images
scale	: 0–do not use scaling/1–use scaling

SHORT DESCRIPTION: The object of the routine is to rotate, translate and scale an image img2 so as to coincide with another (reference) image img1. The correspondence is determined by two sets of feature points feat1x, feat1y, feat2x,

feat2y. The computation is performed in two steps. First, the scaling, translation and rotation are computed from the given feature points. Then, the affine transform of img2 is performed according to the computed parameters. The transformed image is stored in a buffer big enough to ensure that no information is lost. If the flag type is 0, the second step is not performed and the execution of the routine stops here. If the flag is 1, the routine proceeds to the refinement of the feature points. The refinement is performed by finding the optimal displacement for each pair of feature points so that the difference of the windows surrounding them is minimized. The difference is computed as the RMS of the difference of the values of the corresponding pixels in the windows, normalized according to mean and standard deviation in each window. The size of the window within which the comparison happens is given by wcwx,wcwy and the range of the displacement tried for each feature point pair by wdwx,wdwy. After the refinement, the feature point sets are sorted according to the similarity of their surrounding regions and only the `qual` best are used for the following. Then, the scaling, translation and rotation are computed from the refined feature points, and img2 is transformed according to them. The transformed image is stored in a buffer big enough to ensure that no information is lost, and is returned to the calling routine. In all the above, only the pixels for which the corresponding mask i1_mask,i2_mask is set are processed.
RETURN VALUE: 0, -1, -9, -24, -40
SEE ALSO: registr, registr_is, comb_registr

1.3.14 WriteTheImage

Subroutine to construct the final mosaiced image
LIBRARY MODE:

```
int WriteTheImage ( K1, K2, current_k1, current_k2,
                    imag_1_tmp, imag_2_tmp, imag_3_tmp,
                    imagr, imagg, imagb, N, M,
                    Min_X, Max_X, Min_Y, Max_Y, color,
                    WhereIsIt, roitop, roileft )

int     K1, K2;
int     current_k1, current_k2;
image   imag_1_tmp, imag_2_tmp, imag_3_tmp;
image   imagr, imagg, imagb;
int     N, M;
int     Min_X, Max_X, Min_Y, Max_Y;
```

```
int       color;
unit      ** WhereIsIt;
int       roitop, roileft;
```

K1, K2	: the number of images on horizontal and vertical axes
current_k1, current_k2	: the numbers of currently processed images
imag_1_tmp, imag_2_tmp, imag_3_tmp	: temporary input image
imagr, imagg, imagb	: output image
N, M	: the X and Y size of an image
Min_X	: the leftmost coordinate of the image
Max_X	: the rightmost coordinate of the image
Min_Y	: the top coordinate of the image
Max_Y	: the bottom coordinate of the image
color	: the option for color or graylevel images
WhereIsIt	: the transpositions of the origin point of each image with respect to the origin of the final mosaicked image.
roitop, roileft	: the origin point of region of interest

SHORT DESCRIPTION: This function reads each pair of images and constructs the mosaicked image by using the matrix with the transpositions(WhereIsIt). This matrix is the output of the Mosaic function.
RETURN VALUES: 0, -1
SEE ALSO: Mosaic, Re_Order_misimage, Check_misimage

1.3.15 WeightPixel

Subroutine to assign a weight to a pixels with respect to the image or images which contain it.
LIBRARY MODE:

```
int WeightPixel ( K1, K2, N, M, X, Y, G, H, weight, WhereIsIt )

int     K1, K2;
int     N, M;
int     X, Y;
int     G, H;
```

```
float   *weight;
unit    ** WhereIsIt;
```

K1, K2	:	number of images on X and Y axes.
N, M	:	X and Y size of the image.
X, Y	:	the number on X and Y axes of the component image where the given pixel comes from.
G, H	:	the coordinate on X and Y axes for the given pixel, with respect to the Mosaiced image.
weight	:	the weight assigned to the pixel (G,H) with respect to that component image (X,Y).
WhereIsIt	:	the transpositions of the origin point of each image with respect to the origin of the final mosaiced image.

SHORT DESCRIPTION: This subroutine assigns a weight to the pixels with respect to the image which contains it. If the given pixel is not from an overlapping area between different images, this weight is 1. When the pixel is from an overlapping part of 2-4 images, than the values of the overlapping pixels are weighted according to the closest component image to that pixel.
RETURN VALUES: 0
SEE ALSO: Mosaic

Index