8.2 Hardware And Software

Although the implementational aspects of change detection are not the main concern of this project, potential improvements in speed should be considered in order to estimate the feasibility of such a technique being extended for practical use.

The operating speed of a particular algorithm can be increased by writing more efficient code and by using faster hardware. Both approaches could be taken for the change detection. The hierarchical search algorithm described above is a parallel algorithm; at a particular scale all the matches are made independently. By implementing the change detection on a parallel processing machine improvements in speed of the order of the number of processors could be achieved. For example, with 100 processors 3D data might be processed in ≈ 2.5 hours. Further improvements might be possible with the design of specialised integrated circuitry to implement the correlation function.

The software written during the project was not written for speed of operation, but so that the problem could easily be investigated. There would no doubt be improvements that could be made if it were to be written to execute quickly. For example, timings carried out at the Dept. of Medical Physics [1] suggested that the removal of IF statements, used to check that a search was not off the edge of an image, would further increase the speed by a factor of 2. A further suggestion was that the normalised correlation function could be implemented using look-up tables. The necessary statistics could be obtained by preprocessing the images. This would remove the need for many multiplications and hence increase operating speed.

[1] Thanks to Martin Connell

8.3 Summary

In this chapter a faster search strategy for the change detection has been proposed. Timings taken and predictions of expected increase in speed show that for 3D data improved algorithms alone may not be sufficient to make the change detection of practical use. Work may be required on both the hardware and software sides of the implementation to bring the operating time of the change detection below 1 hr. This depends on the speed of serial computation available and how much more efficiently the algorithms can be coded.

Chapter 9

Discussion, Conclusions and Further Work

In this dissertation the problem of change detection in MRI brain scan data using a point-to-point matching approach has been introduced, a solution has been described and tests assessing the performance of this solution have been shown. The results of these tests are discussed in this chapter, conclusions are presented on the achievements of the project and suggestions are made for further work.

9.1 Discussion Of Test Results

The test results shown have demonstrated the performance of each of the individual stages of the change detection and of the change detection as a whole.

The removal of the spot noise was successful and did not alter the data beyond the competence of the rest of the change detection.

Enhancing features by local standard deviation as a measure of expectation of a good match seemed reliable in the light of the performance of the smoothing function.

The initial mapping, which used local normalised correlation, overcame the grey scale shifts between the images of Figs. 1–1 and 7–9. The decision to execute a search for matches by translation alone was validated by the fact that image rotations and magnifications were successfully calculated and removed in Tests 3a,

3b, 4, 5, 6, 7 and 8 of Chapter 7. Spurious matches did occur as expected within the initial mapping and there were also groups of bad matches near edges. It was not clear whether these bad matches were the effects of the magnification and rotation algorithms used to generate the test data or a problem with ambiguous edge matching. Overall the initial mapping operated well, otherwise the transformation parameters would not have been calculated correctly. It remains uncertain as to how much an image can be rotated and scaled before the local normalised correlation and translation search will become useless. A method of formally assessing the technique would enable a cut off point to be set. Only small rotations are expected to occur after the image has been approximately registered at the preprocessing stage.

The smoothing function demonstrates the required behaviour, removing spurious vectors and taking into account neighbourhood trends. The calculation of the weighting by multiplying and normalising the correlation and local standard deviation seems appropriate, the statistical results suggest that the weights were distributed appropriately and the performance of the smoothing supported this. Tests 9a and 9b and the test on real data of Chapter 7 showed how different smoothing parameters could affect the residual mappings. The tests of Chapter 5 go some way to understanding how to choose parameters, but a more rigorous analysis is needed. This is especially true of the case using real data where it is not known what the mapping is expected to look like after smoothing.

The calculation and extraction of the transformation parameters produces accurate estimates, despite some bad matches. The poorer accuracy when larger rotations or rotations and magnifications were tested (Test 3b, 6 and 8 of Chapter 7) is likely to be due to the initial mapping suffering and so not providing such accurate data for the smoothing and hence the transformation extraction stage.

As a whole, the stages of the technique combined well together, but the reliance on the initial mapping was demonstrated by the fact that groups of bad matches propagated through to the residual data.

Test 9 and the test on the real data in Chapter 7 were encouraging. Tests 9 showed the ability of the technique to detect changes in small regions. In the test on real data, despite the distortion and larger changes than would be expected,

the technique was able to overcome the grey scale changes, detect the changes in the ventricles and plausibly reconstruct the ventricles from the residual mapping.

To summarise, the tests carried out suggest that this technique has the potential to both register images and detect changes. The stages of the change detection have been shown to demonstrate the required behaviour and meet the requirements set out in Chapter 3. The tests carried out were all on 2D slices and similar testing would have to be carried out on 3D images to confirm the ability of the algorithms to extend to 3 dimensions.

9.2 Conclusions

This project can be deemed to be a measured success. A new technique for change detection based on point-to-point matching has been investigated and algorithms to effect its operation have been explained and results to indicate the performance have been presented. The implementations of these algorithms are successful as demonstrated by the results. Both the strengths and weaknesses of these algorithms have been discussed and the problems likely to hinder further investigation of such a technique have been identified, namely; computational complexity in 3D and the groups of bad matches that occur on image edges, although tests with non-manufactured, rotationally misaligned MRI images might show the bad matches to be an artifact of the rotation algorithms used to generate test data. It was suggested that the Dept. of Medical Physics would take this work further, but this relied on the ability to overcome the operating speed and bad initial matches.

To increase the confidence with which the smoothing can be used a formal way of deciding the smoothing parameters is required but this would not hinder further work.

It might be hoped that the extension of this project to 3D would find the matching more successful given the fact that there will be more structure in 3D.

The combination of correlation to provide a mapping and smoothing to preserve neighbourhood trends appears to be a new technique and may extend to other image matching applications.

9.3 Further Work

To extend the work carried out during this project further investigations should be carried out into the initial mapping, representation and smoothing and a method of analysing the residual data should be developed. If the technique were to be put to practical use work would be required in improving the speed of operation and a way of assessing the performance of the technique would be needed. These points are discussed below.

9.3.1 Theoretical Aspects

Initial Mapping

The function used to find a match between two regions in the images was a normalised correlation function. Two approaches which could be taken to improve the initial mapping are to apply a different search strategy for finding matches and to investigate different matching functions [Herbin 89]. The problem of bad matches may be overcome by having a symbolic change analysis of the residual data as outlined in Section 6.4, or different search strategies may avoid looking in regions where ambiguous matches could occur.

Representation

As outlined in Section 3.1 the implementation of this technique can only represent contractions satisfactorily. This could be avoided by having an initial mapping from $\mathbf{I}_1$ to $\mathbf{I}_2$ and an initial mapping from $\mathbf{I}_2$ to $\mathbf{I}_1$. The two mappings could then be "parsed" to produce one to many and many to one mappings before being smoothed. Presumably the total number of mappings would remain constant. This approach might reduce spurious matches before smoothing, but could complicate the smoothing process.

Smoothing

The results in Section 5.3 show that the smoothing process behaved approximately as expected. However, before it can be used with confidence its dynamic characteristics need to be more fully understood. Answers to the following questions are required.

i) Given a particular gain and number of iterations over how many pixels does the influence of each vector extend?

ii) Are the gain and number of iterations independent variables or will the same behaviour be displayed by many different combinations of gain and number of iterations?

iii) Is there an underlying rigid neighbourhood model that can help determine a more powerful smoothing algorithm?

Whatever smoothing function is used the extent of influence of vectors and the way in which a neighbourhood affects a central vector must be known. The behaviour of the smoothing can then be predicted and controlled, by choosing an appropriate gain and number of iterations.

Analysis of Residual Data

Section 6.4 contains suggestions for further work on the analysis of the residual data.

9.3.2 Practical Aspects

Speed of Operation

If this change detection technique were to be put to practical use the problem of computational complexity would have to be overcome. Likely investigations would involve testing quicker search strategies, faster implementations of algorithms, the use of parallel computing and possibly the use of specialised hardware to implement certain processes.

Assessment of Performance

This technique always produces results, but how reliable are these results? Some measure of the performance of the technique on sets of images is required. A simple measure of accuracy of the transformation parameter estimates could be given by the variance of the calculated image transformation parameters [Meertens 90].

Order of Processing

The assumed order of operations may not be the best one. Once a method of assessing performance has been developed other orderings could be investigated. For instance; Looking for n good match candidates in the skull, matching these, calculating the image transformation from these n matches and then use the knowledge of the transformation to guide the search for an initial mapping.

If this technique is to be investigated further the main areas of effort should be on reducing the speed of operation of the technique and on eliminating the groups of bad matches from the initial mapping.

Bibliography

[Apicella 89] A. Apicella, J.S. Kippenham, J.H. Nagel. *Fast Multi-modality Image Matching.* **SPIE Vol. 1092 Medical Imaging III: Image Processing (1989), pp 252-263.**

[Bajcsy 83] R. Bajcsy, R. Lieberson, M. Reivich. *A Computerised System for the Elastic Matching of Deformed Radiographic Images to Idealized Atlas Images.* **J. Comp. Assisted Tomography, 7(4), pp 618-625, 1983.**

[Bajcsy 89] R. Bajscy, S. Kovačič. *Multiresolution Elastic Matching.* **Computer Vision, Graphics and Image Processing 46, pp 1-21, 1989.**

[Ballard 82] D.H. Ballard, C.M. Brown. *Computer Vision.* **Prentice-Hall, New Jersey, 1982.**

[Barnard 87] S.T. Barnard. *Stereo Matching by Hierarchical Microcanonical Annealing.* **Proc. 10th Int. Joint Conf. on AI, pp 823-835, 1987.**

[Correia 90] J.A. Correia. *Editorial: Registration of Nuclear Medicine Images.* **The Journal of Nuclear Medicine, Vol. 31, No. 7, pp 1227-1229, July 1990.**

[Dann 88] R. Dann, J. Hoford, S. Kovačič, M. Reivich, R. Bajcsy. *Three-Dimensional Computerized Brain Atlas for Elastic Matching: Creation and Initial Evaluation.* **SPIE Vol. 914 Medical Imaging II(1988), pp 600-608.**

[Fisher] R.B. Fisher. *MSc Machine Vision notes.* **Dept. of Artificial Intelligence, Edinburgh University.**

[Hallam] J. Hallam. *Computational Theories for Research in Vision and Cognition.* **Dept. of Artificial Intelligence, Edinburgh University.**

[Herbin 89] M. Herbin, A. Venot, J.Y. Devaux, E. Walter, J.F. Lebruchec, L. Dubertret, J.C. Rouycarol. *Automated Registration of Dissimilar Images: Application to Medical Imagery.* **Computer Vision, Graphics and Image Processing 47, pp 77-78, 1989.**

[Junck 90] L. Junck, J.G. Moen, G.D. Hutchins, M.B. Brown, D.E. Kuhl. *Correlation Methods for the Centering, Rotation, and Alignment of Functional Brain Images.* **The Journal of Nuclear Medicine, Vol. 31, No. 7, pp 1220-1226, July 1990.**

[Lee 86] B.G. Lee, V.T. Tom, M.J. Carlotto. *A Signal-Symbol Approach To Change Detection.* **Proc. AAAI-86: 5th Nat. Conf. on AI, pp 1138-1143, Aug. 1986.**

[Marr 82] D. Marr. *Vision.* **W.H. Freeman & Co., 1982.**

[Meertens 90] H. Meertens. *Field Placement Errors in Digital Portal Images.* **Physics in Medicine and Biology, Vol. 35, No. 3, p 318, 1990.**

[Pelizzari 89] C.A. Pelizzari, G.T.Y. Chen, D.R. Spelbring, R.R. Weichselbaum, C. Chen. *Accurate Three-Dimensional Registration of CT, PET, and/or MR Images of the Brain.* **J. Comp. Assisted Tomography, 13(1), pp 20-26. Jan./Feb., 1989.**

[Press 89] Press *Numerical Recipes in C: The art of scientific computing.* **Cambridge University Press, 1989.**

[Saeed 89] N. Saeed, S. Marshall, H.S. Young, T.S. Durrani. *A System For Automatic Patient Realignment in MR Imaging of the Head.* **Proc. ICASSP 89, Vol. 3, pp 384-387. (IEEE publication)**

[Tikhonov 77] A.N. Tikhonov, V.Y. Arsenin. *Solutions of Ill-Posed Problems.* **Winston and Sons, 1977.**

[Wells 89] M.G. Wells, A.N.R. Law, A.R. Allen. *An Expert System For The Identification Of Meaningful Regions Within MR Brain Images.* 3rd Int. Conf. Image Processing and its Applications, Warwick, 1989, pp 378-383. (IEE Conf. Publ. No. 307)

Appendix A

Hardware, Software and Displays

A.1 Hardware

The machines used throughout this project were Hewlett Packard 9000/300 series with a maths co-processor. The colour screen used to produce the displays, photographs of which appear throughout the dissertation, was an HP98752A.

A.2 Software

All the code was written in "C" and run under a UNIX operating system. Appendix B contains the code developed to implement the change detection.

A.3 Displays

The colour displays were produced using the X windows programming tools. The pictures were loaded into an Xwidget which was a StaticRasterImage. Associated with each display was a colour map defining the scale of colours.

On entering an X window environment the displays were invoked by typing:

```
display <imagefile> <colourmapfile> [-name 'icon_name'] &
```

The `-name` is an option which associates `'icon_name'` with the icon when the display window is iconised. Thus enabling iconised images to be identified by their name.

See **makedisplay.c** in Appendix B on how to create different types of image for display. The colour map is simply a text file containing 255 lines, each line starts with a '#' and then has 3 pairs of consecutive hex numbers. Each pair of numbers gives the red, green and blue values to be associated with the pixel value at which the line is. Eg:

```
(line 15) #0fabe8
```

This would associate the pixel value of 15 with a red colour of 0f a green colour of ab and a blue colour of e8. Thus by having files containing different colour maps, each display can use different colours to display different information.

The code for display.c is:

```
#include "Xinclude.h" /*Includes all the X functions and headers*/

Display * display;
Widget toplevel,bboard;
Widget rasterimage;

/*********************************************************/
/*                        Widget argument assignment   */
/*********************************************************/

static Arg toplevelArgs [] =
{
        {XtNtitle, (XtArgVal) "IMAGING DATA"},
        {XtNallowShellResize, (XtArgVal) True}
};

static Arg bboardArgs [] =
{
        {XtNlayout,(XtArgVal) XwMINIMIZE}
};

static XImage * image;

static Arg imageArgs [] = {
   {XtNdrawColor, (XtArgVal)0 },
   {XtNeraseColor,(XtArgVal)0 },
   {XtNimage,     (XtArgVal)NULL},
   {XtNgridThickness, (XtArgVal)0 }
 };

/*****************************************************************/
```

```
/*Graphics handler: Changes the colourmap when entering a widget*/
/*****************************************************************/

void gx_handler( w, client_data, event )
Widget w;
caddr_t client_data;
XEvent * event;
{
        XInstallColormap( XtDisplay(w), *(Colormap *)client_data );
}

void main(argc, argv)
int argc;
char *argv[];
{
        pixel *picture;
        Colormap cmap, original;
        struct map_properties *map;
        int height,width,frames;
        int screen,im,count;
        int i;

        picture=imagetomem(argv[1],&width,&height,&frames);

/*****************************************************************/
/*           Create The Top Level Widget                         */
/*****************************************************************/


        if(argc==4 && strcmp(argv[3],"-name") == 0)
        {
           toplevel = XtInitialize (argv[4], "menusample", NULL,
                                                  0, &argc,argv);
        }
        else
        {
           toplevel = XtInitialize (argv[0], "menusample", NULL,
 0, &argc,argv);
        }
        XtSetValues (toplevel,toplevelArgs,XtNumber(toplevelArgs));

        bboard=XtCreateManagedWidget("bboard",XwbulletinWidgetClass,
                        toplevel,bboardArgs,XtNumber(bboardArgs));
            .

        display = XtDisplay( toplevel );
        screen  = DefaultScreen( display );
```

```
        original =  DefaultColormap(display,DefaultScreen( display) );

/*************************************************************/
/*                       Create Raster Image                 */
/*************************************************************/

        image = XCreateImage( display, DefaultVisual(display,screen),
                      DefaultDepth(display,screen),ZPixmap,0,picture,
                                                 width,height, 8,0);
        if (image == NULL){printf("Create image failed\n");exit(1);}

/*************************************************************/
/*                     Put image into StaticRaster Widget      */
/*************************************************************/

        imageArgs[0].name =XtNsRimage;
        imageArgs[0].value=(XtArgVal) image;
        imageArgs[1].name =XtNwidth;
        imageArgs[1].value=(XtArgVal)width;
        imageArgs[2].name =XtNheight;
        imageArgs[2].value=(XtArgVal)height;
        imageArgs[2].name =XtNdepth;
        imageArgs[2].value=(XtArgVal)DefaultDepth(display,screen);

        rasterimage = XtCreateManagedWidget( "asas",
          XwstaticRasterWidgetClass,bboard,imageArgs,
                            XtNumber( imageArgs ) );

        XtAddEventHandler (rasterimage, EnterWindowMask,TRUE,
                        gx_handler, (caddr_t) &cmap );
        XtAddEventHandler (rasterimage, LeaveWindowMask,TRUE,
                        gx_handler, (caddr_t) &original );

/*********************************************************/
/* Realize the widget tree, and start processing events */
/*********************************************************/

   XtRealizeWidget (toplevel);

   if ( NewColormap( display,XtWindow(rasterimage),&cmap)!=0)
      {
      char line[40];
      int pix = 0;
      FILE * fp = fopencheck(argv[2],"r","COLOURMAP");

      while ( fscanf(fp, "%[^\n]\n", line )!= EOF )
         {
```

```
            char * cs = strtok( line, " \n\r\t" );
            while ( cs != NULL )
               {
               XColor col;
               Status res = XParseColor( display, cmap, cs, &col );
               col.flags = DoRed | DoGreen | DoBlue;
               col.pixel = pix++;
               if ( res != 0 )  XStoreColor( display, cmap, &col );
               else printf("CMSMAP error (%d %s)\n",pix, cs );
               cs = strtok( NULL, " \n\r\t" );
               }
            }
         fclosecheck(fp,"COLOURMAP");
         XFlush(display);
         }
      XtMainLoop();
}

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

NewColormap() attempts to install a new un-initialised colormap and
attach it to the specified window. This means whenever the window is
ENTERED the colormap is installed on the hardware, then un-installed
when a LEAVE event is registered.

Subsequently created sub-windows of the specified window will inherit
this colormap under the default inheritance mechanism
(copy-from-parent). Sub-windows of the specified window which were
created before thiscolormap was attached all have their own individual
access to the colormap attached to the parent at their creation time.
This explains how nothing appears to happen when a new colormap is
attached to the root window - only when new direct children of the root
are created and entered does the new colormap  become installed. This
can appear confusing, but makes perfect sense.

Note:- the colormap is created with AllocAll set. This means the
created colormap is PRIVATE to the window.

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

int NewColormap( display, window, cmap )
Display *display;
Window window;
Colormap * cmap;
{
XVisualInfo vinfo;
Status res = XMatchVisualInfo( display, DefaultScreen(display), 8,
```

```
                                                PseudoColor, &vinfo );

if (res ==0)
   {
   fprintf(stderr,"Screen cannot support 8-bit deep PseudoColor
                                                    applications\n");
   return(0);
   }

*cmap = XCreateColormap( display, window, vinfo.visual, AllocAll );

XSetWindowColormap( display, window, *cmap );
return(1);
}
```

Appendix B

"C" Source Code

The operation of the various programs written to implement the algorithms is detailed below. Much of this code would become redundant in a practical change detection system. Since it was written to investigate how the algorithms were operating it offers features which would be superfluous once the investigation was complete.

B.1 Processing Stages

The operations executed to produce a residual mapping of vectors are as follows: (Output files are denoted with a dollar, $, sign.)

B.1.1 Conservative smoothing

To conservatively smooth an image type:

```
spotnoise <imagefilein> <region> <smoothed_image_out>$
```

`<region>` is the name of a text file which defines a region. The format for region definitions, image files and vector mappings is outlined in Section B.2. Both the images to be input to the change detection should be smoothed.

B.1.2 Feature Enhancement

To enhance the features of an image type:

```
enhance <image> <c.reg> <sampling> <SDfile_out>$
```

The limits of the area to be enhanced are then input when requested. The limits, x1,y1,x2,y2, define a rectangular region with its upper left corner at (x1,y1) and lower right in (x2,y2). The sampling rate determines how often to measure the standard deviation of a pixel. Eg. a sampling rate of 4 would sample every 4th pixel. The sampling feature becomes redundant and should be 1 when a full change detection is to be implemented.

`<c.reg>` is a file containing the definition of a region over which to measure the standard deviation.

The `<SDfile_out>` is a file in exactly the same format as a vector mapping, but the local standard deviation is associated with each point instead of the correlation value.

B.1.3 Initial Mapping

To find an initial mapping type:

```
vectmap <image1> <image2> <c.reg> <s.reg> <sampling> <v_map_file_out>$
```

The limits of the area to be matched are input when requested. `<c.reg>` and `<s.reg>` define the correlation and search regions respectively. The correlation region should be the same as the region used to enhance the features. `<v_map_file_out>` is a vector mapping containing the information on the matches made. Its format is explained in Section B.2. The sampling rate and area limits must be the same as those used to enhance the image.

B.1.4 Weights

To calculate the weights type:

```
calcweights <v_map_file_out> <SDfile_out> <weightsout>$
stretch <weightsout> <normalised_weights>$
```

These two operations could be combined into one, but the absolute values of the weightings were of interest, hence the two operations during the investigation.

B.1.5 Smoothing

To smooth the initial mapping type:

```
convert <normalised_weights> <norm_weights>$
smoothmap <norm_weights> <gain> <iterations> <neighbours> <smooth_out>$
reconvert <smooth_out> <smoothed_map_out>$
```

<neighbours> is a file containing the definition of a neighbourhood region. The reason for the conversion was so that the vector mapping could be viewed easily as text. Once again the conversion and reconversion would become redundant in a practical system.

B.1.6 Calculation and Extraction of Transformation Parameters

The final stage is to remove the effects of image transformation:

```
transthresh <image1> <smoothed_map> <threshold> <final_data_out>$
transform <finaldata>$
remove <finaldata> <mx> <my> <tx> <ty> <residuals>$
```

transthresh removes background data from the vector mapping. (This was left in during the initial mapping and smoothing so that the behaviour of noise could be compared to the behaviour of the data in the brain region). <threshold> is a data value below which points are removed. transform estimates the transformation parameters (as described in Chapter 6) and outputs their values (mx,my,tx,ty) to the screen. remove removes the effects of the estimated transformation using the transformation parameters output by transform. The final file, <residuals>,

is a vector mapping indicating where pixels from **<image1>** have been estimated to move to in **<image2>** of the initial mapping. The **<residuals>** file can then be used to produce displays to estimate the performance of the change detection.

B.2 File Formats

B.2.1 Image files

Image files contain a small header stating the width, height and number of frames in an image. Each of these parameters is on a different line. On the fourth line the image data begins. The value of each byte is the value of that pixel in the imge. Eg. The 100th byte from the start of the fourth line might have a value of 30. The 100th pixel in the image would then have a value of 30. The pixels were loaded into a one dimensional array in a continuous stream. The first byte was the left uppermost pixel in the image.

B.2.2 Region Definitions

The region definitions were contained in text files. The first line stated how large the region was. The following lines contained a number. This number stated how many pixels back or forward had to be moved to reach a new pixel in the region.

For example. In an image 128×128 to move up one line would require moving -128 pixels. To move left three would require moving -3 pixels. The definition for a 3×3 region in a 128×128 image is shown below:

```
9
-257
-256
-255
-129
-128
-127
-1
0
```

```
1
127
128
255
256
257
```

B.2.3 Neigbourhood Definitions

The definition of neighbourhoods for smoothing was complicated slightly by the fact that sampled mappings were being smoothed during testing. The neighbourhood files containing the size of the region on the first line and each subsequent line contained the number of sampling steps to take in the x,y and z directions to reach a neighbouring pixel. These sampling steps are converted into a number of pixels to move to find a neighbouring sample by the smoothing program, `smoothing.c`.An example neighbourhood defintion is shown below: (This defines a neighbourhood to be adjacent and diagonally connected samples)

```
8
-1,1,0
0,1,0
1,1,0
-1,0,0
1,0,0
-1,-1,0
0,-1,0
1,-1,0
```

B.2.4 Vector Mappings

A vector mapping file contains textual information on how the mapping was created in a header. The defintion for this header is in `Headers.h`. The following lines each contained the x,y,z position of a pixel, the position of the pixel which had been found to match with it and the weighting associated with that match. An example is shown below, with only a few vector mappings, but the full text header.

```
Image Dimensions, 128 (x) by 128 (y) by 1 (z)
Image ../IMAGES/warp mapped to image ../IMAGES/mri.1nse
Vector count = 13924
Sampling rate, 1
Region mask, ../Region21
Correlation mask, ../Corr5c128
Start of area mapped, (5.000000,5.000000,0.000000)
Finish of area mapped, (122.000000,122.000000,0.000000)
5.000000,5.000000,0.000000,3.000000,15.000000,0.000000,0.671238
6.000000,5.000000,0.000000,14.000000,15.000000,0.000000,0.650620
7.000000,5.000000,0.000000,4.000000,5.000000,0.000000,0.688105
8.000000,5.000000,0.000000,5.000000,5.000000,0.000000,0.647018
9.000000,5.000000,0.000000,14.000000,15.000000,0.000000,0.752074
10.000000,5.000000,0.000000,15.000000,15.000000,0.000000,0.735008
```

B.3 Code Listings

Throughout the course of the project many functions were written to implement the algorithms, access data, provide statistics etc. Only the code required to execute the processing stages above is included here. The code was written with the ease of accessing data at all times in mind. Additionally, some of the functions include a sampling rate, which causes the program to only sample certain pixels across an image. If the change detection technique is to be investigated further it is suggested that this code is rewritten since some of the stages are redundant now that the algorithms have been proven to work. The correlation function of `imfunctions.c` in particular should be reimplemented since the operating speed of this function is a major limiting factor on the speed of the change detection.

B.3.1 Makefile

```
#   Makefile for Change Detection Code


# series 300 C flags:
CFLAGS = -g -Wc,-Nd4000 -Wc,-Ns4000 -Wc,-Nt50000
# series 800 C flags:
```

```
#CFLAGS = -g

LIBS = -lm
ALLOBJS = imageio.o imfunctions.o graphics.o masksio.o vectorio.o
                              fileio.o vectorfns.o conversions.o

.o : .c

convert : $(ALLOBJS) convert.o
        cc $(CFLAGS) -o convert $(ALLOBJS) convert.o $(LIBS)

transform : $(ALLOBJS) transform.o
        cc $(CFLAGS) -o transform $(ALLOBJS) transform.o $(LIBS)

enhance : $(ALLOBJS) enhance.o
        cc $(CFLAGS) -o enhance $(ALLOBJS) enhance.o $(LIBS)

stretch : $(ALLOBJS) stretch.o
        cc $(CFLAGS) -o stretch $(ALLOBJS) stretch.o $(LIBS)

calcweights : $(ALLOBJS) calcweights.o
        cc $(CFLAGS) -o calcweights $(ALLOBJS) calcweights.o $(LIBS)

smoothmap : $(ALLOBJS) smoothmap.o
        cc $(CFLAGS) -o smoothmap $(ALLOBJS) smoothmap.o $(LIBS)

vectmap : $(ALLOBJS) vectmap.o
        cc $(CFLAGS) -o vectmap $(ALLOBJS) vectmap.o $(LIBS)

remove : $(ALLOBJS) remove.o
        cc $(CFLAGS) -o remove $(ALLOBJS) remove.o $(LIBS)

transthresh : $(ALLOBJS) transthresh.o
        cc $(CFLAGS) -o transthresh $(ALLOBJS) transthresh.o $(LIBS)

makedisplay : $(ALLOBJS) makedisplay.o
        cc $(CFLAGS) -o makedisplay $(ALLOBJS) makedisplay.o $(LIBS)

spotnoise : $(ALLOBJS) spotnoise.o
        cc $(CFLAGS) -o spotnoise $(ALLOBJS) spotnoise.o $(LIBS)
```

B.3.2 ImageDefs.h

```
#include <stdio.h>

typedef unsigned char pixel;

struct local_mask
{
        int mask_size;          /* The size of the mask          */
        int *mask_overlay;      /* A pointer to an array which   */
};                              /* defines the mask shape        */

struct offset { int value[3]; };

struct vector
{
        double beg[3];          /* x,y,z start Coordinates       */
        double end[3];          /* x,y,z finish Coordinates      */
};

struct vectormap
{
        struct vector mapping;
        double strength;
};

struct region
{
        int regionsize; /* The size of a region for smoothing */
        struct offset *points;  /*x,y,z dist. from centre of region*/
};

#include "Headers.h"

/* For maths */
double sqrt();
double pow();
double atan();
double atan2();

/* From conversions.c */

int threedtoline();
struct offset linetothreed();
int outofrange();
struct offset counttoxyz();
int outoflimits();

/* From fileio.c */

FILE *fopencheck();
fclosecheck();
```

```
help();

/* From vectorio.c */

vectormaptofile();
vectorheadertofile();
mappingtofile();
struct vectormap *vectormaptomem();
vectorheadertomem();
mappingtomem();
displayvectormapheader();

/* From masksio.c */
struct region *regiontomem();
struct local_mask masktomem();
inputmask();

/* From imfunctions.c */
double correlate();

/* From imageio.c */
pixel *imagetomem();

/* From smoothmap.c */

struct vectormap unnamed();
struct offset arrowplus();
struct vectormap smooth();
double sqrsize();
```

B.3.3 Headers.h

```
# Headers file

struct vectormapheader
{
        int image_dims[3];      /* The x,y & z image dimensions       */
        char imagefrom[20];     /* Name of the image mapped from      */
        char imageto[20];       /* Name of the image mapped to        */
        int vectorcount;        /* The number of vectors in the file  */
        int sampling;           /* The sampling rate                  */
        int corrmask[10];       /* The correlation mask used          */
        int regionmask[10];     /* The region mask used               */
        struct vector area;     /* Top left and bottom right coords of*/
};                              /* area mapped                        */

struct maskheader
{
        int masksize;           /* Number of pixels in mask           */
        int extent[3];          /* The x,y and z distances from the   */
```

```
                                /* centre of the mask.                  */
};
```

B.3.4 calcweights.c

```
/* ------------------------------------------------------------------
Program:        calcweights

Function:       Takes a file containing local standard deviation and a
                vector mapping and calculates unnormalised weightings.
                The output is the old mapping with the weights
                calculated.

Usage:          calcweights <SDfile> <vectmapping> <weightedmapping>

Dependancies:   vectorio.c, fileio.c. - ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */

#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char *argv[];
{
        struct vectormap *varmap,*cormap,*commap;
        struct vectormapheader vhdr,chdr;
        int count;

        varmap = vectormaptomem(argv[1],&vhdr);
        cormap = vectormaptomem(argv[2],&chdr);

        commap =(struct vectormap*)malloc(vhdr.vectorcount*
                                        sizeof(struct vectormap));

        for(count=0;count<vhdr.vectorcount;count++)
        {
                commap[count].mapping=cormap[count].mapping;
                commap[count].strength=cormap[count].strength*
                                        varmap[count].strength;
        }

        vectormaptofile(argv[3],commap,chdr);
}
```

B.3.5 conversions.c

```
/* ----------------------------------------------------------------
Program:        conversions (non-executable)

Function:       When making correspondences between two images only
                certain points might be sampled. The data concerning
                these sampled matches is stored in a one dimensional
                array. The functions provided here enable the
                calculation of the position in the one dimensional array
                of the data for a particular x,y,z point and vice versa.
                To enable this calculation the minima and maxima
                defining the cubic volume within which points were
                sampled and the sampling rate need to be known.

Usage:          Called from other functions only.

Functions:      int threedtoline(pos,min,max,sampling)
                struct offset linetothreed(count,min,max,sampling)

Dependancies:   ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
---------------------------------------------------------------- */

#include <stdio.h>
#include "ImageDefs.h"

/* ----------------------------------------------------------------
Function:       threedtoline(pos,min,max,sampling)

Arguments:      pos - A pointer to an offset structure.
                min - A pointer to an array containing the values of the
                        minimum x,y and z coordinates.
                max - A pointer to an array containing the values of the
                        maximum x,y and z coordinates.
                sampling - How often pixels were sampled (ie 1 every 4 =
                        sampling value of 4)

Returns:        An integer equivalent to the value which the data
                sampled at pos would occupy in a one dimensional array,
                starting at 0, containing all the sampled data.
---------------------------------------------------------------- */

int threedtoline(pos,min,max,sampling)
int min[3],max[3],sampling;
struct offset pos;
{
        int count,xval,yval,zval,width,height;

        width=((max[0]-min[0])/sampling)+1;
        height=((max[1]-min[1])/sampling)+1;
```

```
        xval=(pos.value[0]-min[0])/sampling;
        yval=((pos.value[1]-min[1])/sampling)*width;
        zval=((pos.value[2]-min[2])/sampling)*width*height;

        count=xval+yval+zval;

        return(count);
}

/* ------------------------------------------------------------------
Function:       linetothreed(count,min,max,sampling)

Arguments:      count - An integer, representing a position in a 1D
                        array.
                min - A pointer to an array containing the values of the
                        minimum x,y and z coordinates.
                max - A pointer to an array containing the values of the
                        maximum x,y and z coordinates.
                sampling - How often pixels were sampled (ie 1 every 4 =
                        sampling value of 4)

Returns:        An offset struct which is the equivalent x,y,z
                coordinate of the sampled data, given the min and max
                values and the sampling rate.
------------------------------------------------------------------ */

struct offset linetothreed(count,min,max,sampling)
int count,min[3],max[3],sampling;
{
        struct offset pos;
        int xval,yval,zval,width,height;

        width=((max[0]-min[0])/sampling)+1;
        height=((max[1]-min[1])/sampling)+1;

        zval=count/(width*height);
        yval=(count-zval*width*height)/width;
        xval=count-zval*width*height-yval*width;

        pos.value[0]=xval*sampling+min[0];
        pos.value[1]=yval*sampling+min[1];
        pos.value[2]=zval*sampling+min[2];

        return(pos);
}

/* ------------------------------------------------------------------
Function:       outofrange

Arguments:      arr - An offset structure, represents point in 3D space.
                min - A pointer to an array containing the minimum x,y
```

```
                    and z coordinates.
                max - A pointer to an array containing the maximum x,y
                    and z coordinates.

Returns:        1 if out of range else 0.
---------------------------------------------------------------- */

int outofrange(arr,min,max)
struct offset arr;
int min[3],max[3];
{
        int axis,flag;
        flag=0;

        for(axis=0;axis<3;axis++)
        {
                if(arr.value[axis]<min[axis] || arr.value[axis]>
                                                        max[axis])
                {
                        flag=1;
                }
        }
        return(flag);
}

/* ----------------------------------------------------------------
Function:       counttoxyz

Arguments:      count - The number of a pixel in an image
                width - The width of the image
                height - The height of the image

Returns:        An offset structure representing the x,y,z position of
                the pixel.
---------------------------------------------------------------- */

struct offset counttoxyz(count,width,height)
int count,width,height;
{
                struct offset coords;
                int x,y,z;

                z=count/(width*height);
                y=(count-z*width*height)/width;
                x=count-z*width*height-y*width;

                coords.value[0]=x;
                coords.value[1]=y;
                coords.value[2]=z;

                return(coords);
}
```

B.3.6 re-convert.c

```
/* ------------------------------------------------------------------
Program:        convert

Function:       Converts a vector mapping from points matching points to
                points with a vector, pointing to the matching point.
                i.e x1,y1,z1,x2,y2,z2,weighting becomes
                    x1,y1,z1,(x2-x1),(y2-y1),(z2-z1),weighting.

Usage:          convert <ip/file> <op/file>

Functions:      NONE

Dependancies:   vectorio.c, fileio.c - ImageDefs.h

Comments:       This conversion is not really necessary if the change
                detection is to be run in one step. The conversion was
                simply so that a data file could be more easily
                inspected and the effects of the smoothing seen, when it
                was under development.

                RECONVERT JUST REVERSES THE CONVERSION PROCESS
                CODE NOT SHOWN

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */

#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char ** argv;
{
        struct vectormap *coordmap;
        struct vectormapheader coordmaphdr;
        int count;

        coordmap = vectormaptomem(argv[1],&coordmaphdr);

        fprintf(stdout,"Conversion starting\n");

        for(count=0;count<(coordmaphdr.vectorcount);count++)
        {
                int axis;
                for(axis=0;axis<3;axis++)
                {
                        double coordval;
                        coordval = coordmap[count].mapping.end[axis];
                        coordmap[count].mapping.end[axis]=
                             coordval-coordmap[count].mapping.beg[axis];
```

```
                }
        }

        fprintf(stdout,"Conversion complete\n");

        vectormaptofile(argv[2],coordmap,coordmaphdr);
}
```

B.3.7 enhance.c

```
/* ------------------------------------------------------------------
Program:        enhance

Function:       Calculates the standard deviation for regions around
                pixels. The output is a vector mapping with the vectors
                blank, but the strength field contains the value of the
                local standard deviation.

Usage:          enhance <image> <region> <sampling> <output>

Comments:       This is an adapted version of vectmap.c and follows
                exactly the same pattern. The output is the same form as
                a vector mapping file, but the standard deviation is in
                the position normally occupied by the weighting.

Dependancies:   imagio.c, fileio.c, masksio.c, vectorio.c - ImageDefs.h


Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */



#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char **argv;
{
        pixel *image1;
        struct local_mask corrmask;
        struct vectormap *matches;
        struct vectormapheader vectheader;
        int *postomatch,regcount,bestpos,newpos,count,regsize;
        int x1,x2,y1,y2,width,height,pos,sampling,samplecount,frames;
        double variance,xbeg,ybeg,zbeg,xfin,yfin,zfin;
        double value,sumsqr,sum,mean,topval;
        int xstart,ystart,xend,yend;
```

```
        fprintf(stdout,"Correlation testing\n");
        sscanf(argv[3],"%d",&sampling);

        fprintf(stdout,"Input area delimiters\n");
        scanf("%d,%d,%d,%d\n",&xstart,&ystart,&xend,&yend);
        fprintf(stdout,"Area defined by - ");
        fprintf(stdout,"%d %d %d %d\n",xstart,ystart,xend,yend);

        samplecount =(xend/sampling-(xstart-1)/sampling)*(yend/sampling-
                                                    (ystart-1)/sampling);

        postomatch=(int *)malloc(samplecount*sizeof(int));
        matches=(struct vectormap *)malloc(samplecount*
                                            sizeof(struct vectormap));

        image1=imagetomem(argv[1],&width,&height,&frames);

        corrmask=masktomem(argv[2]);

        pos = 0;

/* Find which pixels to sample */

        for(count=0;count<width*height;count++)
        {
                if(count%width<xstart || count%width>xend)
                        continue;
                if(count/width<ystart || count/width>yend)
                        continue;
                if(count%sampling>0)
                        continue;
                if((count/width)%sampling>0)
                        continue;
                postomatch[pos++]= count;
        }

        printf("Samplecount = %d ,(%d)\n",samplecount,pos);

        regsize = corrmask.mask_size;

        for(count=0;count<pos;count++)
        {
                struct vector vtemp;

                mean=0;
                sum=0;
                sumsqr=0;

                if (count%100 == 0)
                {
                        printf("Reached %d\n",count);
                        fflush(stdout);
```

```
        }

        if(postomatch[count] < 0 || postomatch[count] > width*height-1)
        {
                printf("Out of range error \n");
                printf("Count = %d\n",count);
                printf("Position = %d\n",postomatch[count]);
                exit(0);
        }
        for(regcount=0;regcount<regsize;regcount++)
        {
                newpos=postomatch[count]+
                                corrmask.mask_overlay[regcount];
                if(newpos < 0 || newpos > width*height-1)
                {
                        printf("Out of range error \n");
                        printf("Count = %d\n",count);
                        printf("Position = %d\n",postomatch[count]);
                        exit(0);
                }
                value=(double)image1[newpos];
                sumsqr = sumsqr + value*value;
                sum = sum + 2*value;
                mean = mean+value/(double)regsize;
        }
        topval = sumsqr - mean*sum+mean*mean*(double)regsize;
        variance=sqrt(topval/(double)regsize);

        x1 = (postomatch[count]%width);
        y1 = (postomatch[count]/width);
        x2 = 0;
        y2 = 0;

        vtemp.beg[0]=x1;
        vtemp.beg[1]=y1;
        vtemp.beg[2]=0;
        vtemp.end[0]=x2;
        vtemp.end[1]=y2;
        vtemp.end[2]=0;

        matches[count].mapping=vtemp;
        matches[count].strength= variance;
}

printf("Made it\n");
vectheader.image_dims[0] = width;
vectheader.image_dims[1] = height;
vectheader.image_dims[2] = 1;
strcpy(vectheader.imagefrom,argv[1]);
strcpy(vectheader.imageto,argv[2]);
vectheader.vectorcount = samplecount;
vectheader.sampling = sampling;
```

```
        strcpy(vectheader.corrmask,argv[3]);
        strcpy(vectheader.regionmask,argv[4]);

        xbeg = matches[0].mapping.beg[0];
        ybeg = matches[0].mapping.beg[1];
        zbeg = matches[0].mapping.beg[2];
        xfin = matches[samplecount-1].mapping.beg[0];
        yfin = matches[samplecount-1].mapping.beg[1];
        zfin = matches[samplecount-1].mapping.beg[2];

        setvector(&(vectheader.area),xbeg,ybeg,zbeg,xfin,yfin,zfin);

        vectormaptofile(argv[4],matches,vectheader,"w");

        free(postomatch);
        free(image1);
        free(corrmask.mask_overlay);
        free(matches);

        return 0;
}
```

B.3.8 fileio.c

```
/* ------------------------------------------------------------------
Program:        fileio (Not executable)

Function:       Provides a check on file opening and closing

Usage:          Called from other code only

functions:      FILE *fileopencheck(filename,option,filetype)
                fileclosecheck(filepointer,filetype)

Written:        Nick Woodward, MSc 1989-1990.
------------------------------------------------------------------ */

#include <stdio.h>

/* ------------------------------------------------------------------
Function:       fopencheck(filename,option,filetype)

Arguments:      filename - The name of the file to be opened
                option - "r","w","a" or "rw"
                filetype - The name of the type of file to be opened.
                        Used for messaging only.

Returns:        A filepointer to the appropriate file.

Side FX:        Sends appropriate messages to standard output.
```

```
Comments:       Exits if file opening is unsuccessful
---------------------------------------------------------------- */

FILE *fopencheck(filename,option,filetype)
char *filename,*option,*filetype;
{
        FILE *fp;

        if(fp = fopen(filename,option))
        {
                fprintf(stdout,"%s file, %s, opened sucessfully\n",
                                                filetype,filename);
        }
        else
        {
                fprintf(stderr,"ERROR : File, %s, cannot be opened\n"
                                                        ,filename);
                exit(1);
        }
        return(fp);
}

/* ----------------------------------------------------------------
Function:       fclosecheck(filename,filetype)

Arguments:      filename - The name of the file to be closed
                filetype - The name of the type of file to be opened.
                        Used for messaging only.

Returns:        VOID

Side FX:        Sends appropriate messages to standard output.

Comments:       Exits if file closing is unsuccessful
---------------------------------------------------------------- */

fclosecheck(filepointer,filetype)
FILE *filepointer;
char *filetype;
{
        if(fclose(filepointer) == 0)
        {
                fprintf(stdout,"%s file closed successfully\n",filetype);
        }
        else
        {
                fprintf(stderr,"ERROR : %s file cannot be closed\n",filetype);
                exit(1);
        }
}
```

B.3.9 graphics.c

```
/* ------------------------------------------------------------------
Program:        graphics

Function:       A simple function to draw lines on an image.

Usage:          Called from other code only.

------------------------------------------------------------------
Function:       drawline

Arguments:      x1,y1 - The start of the line.
                x2,y2 - The finish of the line.
                image - A pointer to the image being drawn in.
                width - The width of the image.
                col - The pixel value (colour) to assign to line points.

Returns:        VOID

Side FX:        The image has a line drawn on it.

Comments:       The line width is one pixel. Experience suggests that
                images need to be enlarged at least 7 times for the
                data in the image to be seen clearly when many lines
                have been drawn. Eg. in a vector mapping display.
------------------------------------------------------------------

Dependancies:   Needs the maths library - ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */

#include "ImageDefs.h"

drawline(x1,y1,x2,y2,image,width,col)
int x1,y1,x2,y2,col;
pixel *image;
{
        double hypoteneuse,adjacent,opposite;
        int count,newx,newy,xsq,ysq;

        xsq = (x2 - x1)*(x2 - x1);
        ysq = (y2 - y1)*(y2 - y1);

        hypoteneuse = sqrt((float)xsq+(float)ysq);

        adjacent = x2 - x1;
        opposite = y2 - y1;

        if(hypoteneuse > 1)
        {
```

```
                for(count=0;count< (int)hypoteneuse ;count++)
                {
                        newx = adjacent/hypoteneuse*count + x1;
                        newy = opposite/hypoteneuse*count + y1;
                        image[newx+width*newy] = col;
                }
        }
        image[x2+y2*width]= 1;
        image[x1+y1*width]= 0;
}
```

B.3.10 imageio.c

```
/* ------------------------------------------------------------------
Program:        imagio (Not executable)

Function:       Provides functions for the input, output and manipulation
                of images.

Usage:          Called from other code only

Functions:      readheader()
                writeheader()
                pixel *imagetomem()
                imagetofile()
                concatimage()
                enlarge()
                decrease()
                overlayimage()

Dependancies:   fileio.c - ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */

#include "ImageDefs.h"

/* ------------------------------------------------------------------
Functions:      readheader & writeheader

Arguments:      filepointer and
                (readheader) pointers to the width, height and frames
                (writeheader) values of the width,height and frames

Returns:        VOID

Side FX:        (readerheader) Assigns the width, height and frames by
                reading the values from the first three lines of an
                image.
                (writeheader) Writes the value of the width height and
```

```
                frames to file.

Comments:       The first three lines of the image files used contain
                the widht, height and number of frames. Lines 4 onwards
                contain the image data. Each byte representing one pixel
                value.
------------------------------------------------------------------ */

readheader(filepointer,widthpntr,heightpntr,framespntr)
FILE *filepointer;
int *widthpntr,*heightpntr,*framespntr;
{
        fprintf(stdout,"Reading header\n");
        fscanf(filepointer,"%d\n",widthpntr);
        fscanf(filepointer,"%d\n",heightpntr);
        fscanf(filepointer,"%d\n",framespntr);
        fprintf(stdout,"Read header\n");
}

writeheader(filepointer,width,height,frames)
FILE *filepointer;
int width,height,frames;
{
        fprintf(stdout,"Writing header\n");
        fprintf(filepointer,"%d\n",width);
        fprintf(filepointer,"%d\n",height);
        fprintf(filepointer,"%d\n",frames);
        fprintf(stdout,"Written Header\n");
}

/* ------------------------------------------------------------------
Functions:      imagetomem and imagetofile

Arguments:      Name of the file and
                (imagetomem) Pointers to the width height and frames.
                (imagetofile) A pointer to the image and the values of
                the width, height and frames.

Returns:        A pointer to an image (imagetomem)
                VOID (imagetofile)

Side FX:        (imagetomem) Assigns enough memory to contain the image
                and loads the image into this memory.
                (imagetofile) Writes an image to file.

Comments:       pixel is set by typedef in ImageDefs.h
------------------------------------------------------------------ */

pixel *imagetomem(imagename,widthpntr,heightpntr,framespntr)
char *imagename;
int *widthpntr,*heightpntr,*framespntr;
{
```

```
        FILE *filepointer;
        int count,width,height,frames;
        pixel pixval,*image;

        filepointer=fopencheck(imagename,"r","IMAGE");

        readheader(filepointer,widthpntr,heightpntr,framespntr);
        width = (*widthpntr);
        height = (*heightpntr);
        frames = (*framespntr);

        fprintf(stdout,"Width = %d; Height = %d; Frames =%d\n",width,height,
                                                                frames);

        image=(pixel *)malloc(width*height*frames*sizeof(pixel));

        for(count=0;count<width*height*frames;count++)
        {
                pixval = (pixel)fgetc(filepointer);
                image[count] = pixval;
        }
        fclosecheck(filepointer,"IMAGE");

        return(image);
}

imagetofile(imagename,mempointer,width,height,frames)
char *imagename;
pixel *mempointer;
int width,height,frames;
{
        FILE *filepointer;
        int count;

        filepointer=fopencheck(imagename,"w","IMAGE");

        writeheader(filepointer,width,height,frames);

        for(count=0;count<width*height*frames;count++)
        {
                fputc((char)mempointer[count],filepointer);
        }
        fclosecheck(filepointer,"IMAGE");
}

/* ---------------------------------------------------------------
Function:       concatimage

Arguments:      lpointer - Pointer to the image to go on the left.
                rpointer - Pointer to the image to go on the right.
                width1 - The width of the left image.
                width2 - The width of the right image.
```

```
                height - The height of the images.

Returns:        VOID

Side FX:        Takes two images (which must be the same height) and
                concatenates them side by side. The conpointer points to
                the concatenated image.

Comments:       Memory must be allocated before the concatenation takes
                place. Used to build displays.
---------------------------------------------------------------- */

concatimage(lpointer,rpointer,conpointer,width1,width2,height)
char lpointer[],rpointer[],conpointer[];
int height,width1,width2;
{
        int cnt1,cnt2,xcount,ycount,totalcnt;
        totalcnt = cnt1 = cnt2 = 0;

        printf("Concatenating\n");

        for(ycount=0;ycount < height;ycount++)
        {
                for(xcount=0;xcount< width1;xcount++)
                {
                        conpointer[totalcnt++]=lpointer[cnt1++];
                }
                for(xcount=0;xcount< width2;xcount++)
                {
                        conpointer[totalcnt++]=rpointer[cnt2++];
                }

        }
        printf("Concatenation complete\n");
}

/* ----------------------------------------------------------------
Functions:      enlarge and decrease.

Arguments:      pointers to an image and an enlarged image and values of
                the width and height, the scaling factor. Call is
                redundant.

Returns:        VOID

Side FX:        enlarge and decrease images.

Comments:       Memory must be allocated for the new images before
                scaling takes place. Used to build displays.
---------------------------------------------------------------- */

enlarge(picture,newpicture,width,height,scale,call)
```

```
pixel picture[],newpicture[];
int width,height,scale;
char *call;
{
        int row,col,count,xpos,ypos;
        pixel value;

        fprintf(stdout,"Enlarging Image\n");

        for(count=0;count<width*height;count++)
        {
                value = picture[count];
                xpos = count%width;
                ypos = count/width;
                for(col=0;col<scale;col++)
                        for(row=0;row<scale;row++)
                        {
                                int newx,newy;
                                newx = xpos*scale+col;
                                newy = (ypos*scale+row)*width*scale;
                                newpicture[newx+newy]=value;
                        }
        }
        fprintf(stdout,"Enlarged\n");
}

decrease(oldpicture,newpicture,width,height,scale)
pixel *oldpicture,*newpicture;
int width,height,scale;
{
        int xcoord,ycoord;
        int localx,localy,newpos,oldpos;
        pixel localmean;


        printf("decreasing image\n");

        localmean = 0;

        for(xcoord=0;xcoord<width;xcoord += scale)
        {
           for(ycoord=0;ycoord<height;ycoord += scale)
           {
              localmean = 0;
              for(localx=0;localx<scale;localx++)
              {
                for(localy=0;localy<scale;localy++)
                {
                   oldpos = (xcoord+localx)+(ycoord+localy)*width;
                   localmean += oldpicture[oldpos]/scale/scale;
                }
              }
```

```
                newpos = xcoord/scale + ycoord/scale*width/scale;
                newpicture[newpos]=localmean;
            }
        }
        printf("image decreased\n");

}

/* ------------------------------------------------------------
Function:       overlayimage

Arguments:      image1 - Pointer to an image.
                image2 - Pointer to an image.
                newimage - Pointer to a blank image.
                width - The width of the images.
                hoeght - The height of the images.

Returns:        VOID

Side FX:        Lays one image over the other by dividing pixel values
                by four and interlacing the new values.

Comments:       Memory must be allocated for the new image before
                overlay occurs. Used to display two images on top of
                each other. The appropriate colourmap must be used when
                displaying such images.
------------------------------------------------------------ */

overlayimage(image1,image2,newimage,width,height)
pixel *image1,*image2,*newimage;
int width,height;
{
        int count;

        for(count = 0;count<width*height;count++)
        {
                pixel val1,val2;

                val1 = image1[count]/16;
                val2 = 16*((int)image2[count]/16);
                newimage[count] = val1 + val2;
        }
}
```

B.3.11 imfunctions.c

```
/* ------------------------------------------------------------
Program:        imfunctions

Function:       Functions to manipulate image data.
```

```
Usage:          Called from other code only

Functions:      double correlate()
                remspotnoise()

Dependancies:   conversions.c - ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
---------------------------------------------------------------- */

#include "ImageDefs.h"

/* ----------------------------------------------------------------
Function:       correlate

Arguments:      pos1 - The (pos1)th pixel in
                image1 - a pointer to an image.
                pos2 - The (pos2)th pixel in
                image2 - a pointer to an image.
                mask - A mask for the correlation region.
                width - The image width.
                height - The image height.

Returns:        A double which is the value of normalised correlation
                when

Comments:       Much faster implementations should be written.
---------------------------------------------------------------- */

double correlate(pos1,image1,pos2,image2,mask,width,height)
pixel *image1,*image2;
int pos1,pos2,width,height;
struct local_mask mask;
{
        double corr_val;
        double mean1,mean2,sum12,sum1,sum2,sqr1,sqr2,top,den,den1,den2;
        int num,*mask_pos,count,newpos1,newpos2,min[3],max[3];

        num = 1;
        mask_pos = mask.mask_overlay;
        mean1 = mean2 = 0;
        sum12 = sum1 = sum2 = 0;
        sqr1 = sqr2 = 0;
        min[0]=min[1]=min[2]=0;
        max[0]=width-1;max[1]=height;max[2]=0;

        for(count=0;count<mask.mask_size;count++)
        {
                struct offset coord1,coord2;

                newpos1=pos1+mask_pos[count];
```

```
                newpos2=pos2+mask_pos[count];

                coord1=counttoxyz(newpos1,width,height);
                coord2=counttoxyz(newpos2,width,height);

                if(!outofrange(coord1,min,max) && !outofrange(coord2,min,max))
                {
                        sqr1 += (int)image1[newpos1]*(int)image1[newpos1];
                        sqr2 += (int)image2[newpos2]*(int)image2[newpos2];
                        sum12 +=(int)image1[newpos1]*(int)image2[newpos2];
                        sum1 += (int)image1[newpos1];
                        sum2 += (int)image2[newpos2];
                        num=num+1;
                }
        }
        mean1 = (double)sum1/(double)num;
        mean2 = (double)sum2/(double)num;
        top = sum12 - sum1*mean2 - sum2*mean1 + num*mean1*mean2;
        den1 = sqr1 + (double)num*mean1*mean1 - 2*mean1*sum1;
        den2 = sqr2 + (double)num*mean2*mean2 - 2*mean2*sum2;
        den = sqrt(den1*den2);

        if(den == 0)
                corr_val = 0;
        else corr_val = (double)top/den;
        return(corr_val);
}

/* ------------------------------------------------------------
Function:       remspotnoise

Arguments:      image - A pointer to an image.
                imageout - A pointer to where the smooth image goes.
                mask - The mask for smoothing.


Returns:        VOID

Side FX:        The smoothed image is placed at the imageout pointer.

Comments:       Memory must be allocated for the smoothed image before
                smoothing takes place.
------------------------------------------------------------ */

remspotnoise(image,imageout,mask,width,height)
pixel *image,*imageout;
struct local_mask mask;
int width,height;
{
        int count,regcount,maskwidth,maskheight,masksize,*postomatch;
        pixel spotval,max,min;
```

```
maskwidth=maskheight=(int)sqrt(mask.mask_size);
masksize=mask.mask_size;
postomatch=mask.mask_overlay;

for(count=0;count<width*height;count++)
{
        if(image[count]<(int)20)
        {
                image[count]=20;
        }
}

for(count=0;count<width*height;count++)
{
        max = 0;
        min = 255;

        if(count%width < (maskwidth - 1)/2)
        {
                imageout[count]=image[count];
                continue;
        }
        if(width - count%width - 1< (maskwidth -1)/2)
        {
                imageout[count]=image[count];
                continue;
        }
        if(count/width < (maskheight -1)/2)
        {
                imageout[count]=image[count];
                continue;
        }
        if(height - count/width - 1< (maskheight -1)/2)
        {
                imageout[count]=image[count];
                continue;
        }
        for(regcount=0;regcount<masksize;regcount++)
        {
                pixel nearval;
                int newpos;

                if(postomatch[regcount] != 0)
                {
                        newpos = count + postomatch[regcount];
                        nearval = image[newpos];
                        if(nearval < min)
                                min = nearval;
                        if(nearval > max)
                        max = nearval;
                }
        }
```

```
                spotval = image[count];

                if(spotval < min)
                {
                        spotval=min;
                }
                if(spotval > max)
                {
                        spotval=max;
                }
                imageout[count]=spotval;

        }
}
```

B.3.12 makedisplay.c

```
/* ------------------------------------------------------------------
Program:        makedisplay

Function:       Creates various displays for use with the display
                command.

Usage:          Has many options. The usage is stated in the code.

Options:        Overlay two images. (-diff)
                Display residual vectors on an image (-vectresid)
                Display vectors with the weight associated with the
                vector affecting the brightness (-vect)
                Display an image and to its right display the pixel
                values determined by the weights of the vectors
                (-strength)
                Display the right hand picture of the -strength option
                on its own. (-strdisp)
                Display on the left image1, in the middle the image
                created by moving pixels in image1 to where the
                residuals map them and on the right display the image
                with which image1 was mathced.(-motion)
                Display the central image of the -motion option on its
                own. (-motdisp)
                Lay two images side by side (-concat)
                All the above options allow scaling to take place as
                well.
                With no option chosen a single image is enlarged.

                Only integer scaling of 1 or greater can occur.

Dependancies:   vectorio.c, graphics.c, imageio.c, fileio.c -
ImageDefs.h
```

```
Written:        Nick Woodward, MSc 1989-1990
---------------------------------------------------------------- */

#include <stdio.h>
#include "ImageDefs.h"

void main(argc, argv)
int argc;
char *argv[];
{
        pixel *picture;
        int scale,height,width,frames,rastwidth,rastheight;
        int count;

        sscanf(argv[1],"%d",&scale);

/* ----------------------------------------------------------------
Option:         -diff

Usage:          makedisplay <sc> <image1> -diff <image2> <displayout>

Comments:       The pixel values from the two image are interlaced.
                See imageio.c for how the images are interlaced,
                function overlay.

Colour map:     pixval(im1)/16 + int(pixval(im1)/16)*16
---------------------------------------------------------------- */
        if(strcmp(argv[3],"-diff") == 0)
        {
                pixel *temp1,*temp2,*temp3;
                temp3 = (pixel *)malloc(width*height*sizeof(pixel));
                picture = (pixel *)malloc(width*height*scale*scale*
                                                    sizeof(pixel));
                temp1=imagetomem(argv[2],&width,&height,&frames);
                temp2=imagetomem(argv[4],&width,&height,&frames);
                overlayimage(temp1,temp2,temp3,width,height);
                enlarge(temp3,picture,width,height,scale,"");
                rastwidth = width*scale;
                rastheight = height*scale;
                free(temp1);
                free(temp2);
                free(temp3);
                imagetofile(argv[8],picture,rastwidth,rastheight,frames);
        }
/* ----------------------------------------------------------------
Option:         -vectresid

Usage:          makedisplay <sc> <image1> -vectresid <mapping> <displayout>

Colour map:     Image 0-254 residual vector 255
```

```
---------------------------------------------------------------- */
        if(strcmp(argv[3],"-vectresid") == 0)
        {
                pixel *temp1;
                struct vectormap *matchpointer;
                struct vectormapheader vectmapheader;
                int x1,y1,x2,y2;

                fprintf(stdout,"Making a residual map image\n");
                temp1=imagetomem(argv[2],&width,&height,&frames);
                picture = (pixel *)malloc(width*height*scale*scale*
                                                        sizeof(pixel));
                for(count=0;count<width*height*frames;count++)
                        if(temp1[count]==255)
                        temp1[count]=temp1[count]-1;
                enlarge(temp1,picture,width,height,scale,"one");
                matchpointer=vectormaptomem(argv[4],&vectmapheader);
                for(count=0;count<vectmapheader.vectorcount;count++)
                {
                        x1 = scale *(matchpointer[count].mapping).beg[0];
                        y1 = scale *(matchpointer[count].mapping).beg[1];
                        x2 = scale *(matchpointer[count].mapping).end[0];

                        drawline(x1,y1,x2,y2,picture,width*scale,255);
                }
                rastwidth = width*scale;
                rastheight = height*scale;
                free(temp1);
                imagetofile(argv[5],picture,rastwidth,rastheight,frames);
        }
/* ----------------------------------------------------------------
Option:         -vect

Usage:          makedisplay <sc> <image1> -vect <vectmap> <displayout>

Colour map:     Image 0-127, Vector 128-255.

---------------------------------------------------------------- */
        if(strcmp(argv[3],"-vect") == 0)
        {
                pixel *temp1;
                struct vectormap *matchpointer;
                struct vectormapheader vectmapheader;
                int x1,y1,x2,y2;

                fprintf(stdout,"Making a vector map image\n");
                temp1=imagetomem(argv[2],&width,&height,&frames);
                picture = (pixel *)malloc(width*height*scale*scale*
                                                        sizeof(pixel));
                for(count=0;count<width*height*frames;count++)
                        temp1[count]=temp1[count]/2;
                enlarge(temp1,picture,width,height,scale,"one");
```

```
                matchpointer=vectormaptomem(argv[4],&vectmapheader);
                for(count=0;count<vectmapheader.vectorcount;count++)
                {
                        int linecol;
                        x1 = scale *(matchpointer[count].mapping).beg[0];
                        y1 = scale *(matchpointer[count].mapping).beg[1];
                        x2 = scale *(matchpointer[count].mapping).end[0];

                        linecol=128+matchpointer[count].strength*127;
                        drawline(x1,y1,x2,y2,picture,width*scale,linecol);
                }
                rastwidth = width*scale;
                rastheight = height*scale;
                free(temp1);
                imagetofile(argv[5],picture,rastwidth,rastheight,frames);
        }
/* ------------------------------------------------------------------
Option:         -strength

Usage:          makedisplay <sc> <im1> -strength <map> <im2> <dispout>

Colour map:     Image 0-255
------------------------------------------------------------------ */
        if(strcmp(argv[3],"-strength") == 0)
        {
                pixel *temp1,*temp2,*temp3;
                struct vectormap *matchpointer;
                struct vectormapheader vectmapheader;
                int x1,y1,z1,position;
                double value,pixval;

                fprintf(stdout,"Making a strength of match image\n");
                temp1=imagetomem(argv[2],&width,&height,&frames);
                picture = (pixel *)malloc(2*width*height*scale*scale*
                                                        sizeof(pixel));
                temp2 = (pixel *)malloc(width*height*sizeof(pixel));
                temp3 = (pixel *)malloc(2*width*height*sizeof(pixel));

                matchpointer=vectormaptomem(argv[4],&vectmapheader);
                for(count=0;count<vectmapheader.vectorcount;count++)
                {
                        x1 = (matchpointer[count].mapping).beg[0];
                        y1 = (matchpointer[count].mapping).beg[1];
                        z1 = (matchpointer[count].mapping).beg[2];
                        position=x1+y1*width+z1*width*height;
                        value=(matchpointer[count].strength);
                        pixval=value*255;
                        temp2[position]=(pixel)pixval;
                }
                concatimage(temp1,temp2,temp3,width,width,height);
                enlarge(temp3,picture,2*width,height,scale,"one");
                rastwidth = 2*width*scale;
```

```
                rastheight = height*scale;
                imagetofile(argv[5],picture,rastwidth,rastheight,frames);
        }
/* ---------------------------------------------------------------
Option:         -strdisp

Usage:          makedisplay <sc> <im1> -strdisp <map> <dispout>

Colour map:     Image 0-255
--------------------------------------------------------------- */
        if(strcmp(argv[3],"-strdisp") == 0)
        {
                pixel *temp1,*temp2;
                struct vectormap *matchpointer;
                struct vectormapheader vectmapheader;
                int x1,y1,z1,position;
                double value,pixval;

                fprintf(stdout,"Making a strength of match image\n");
                temp1=imagetomem(argv[2],&width,&height,&frames);
                picture = (pixel *)malloc(2*width*height*scale*scale*
                                                        sizeof(pixel));
                temp2 = (pixel *)malloc(width*height*sizeof(pixel));

                matchpointer=vectormaptomem(argv[4],&vectmapheader);
                for(count=0;count<vectmapheader.vectorcount;count++)
                {
                        x1 = (matchpointer[count].mapping).beg[0];
                        y1 = (matchpointer[count].mapping).beg[1];
                        z1 = (matchpointer[count].mapping).beg[2];
                        position=x1+y1*width+z1*width*height;
                        value=(matchpointer[count].strength);
                        pixval=value*255;
                        temp2[position]=(pixel)pixval;
                }
                enlarge(temp2,picture,2*width,height,scale,"one");
                rastwidth = width*scale;
                rastheight = height*scale;
                imagetofile(argv[5],picture,rastwidth,rastheight,frames);
        }
/* ---------------------------------------------------------------
Option:         -motion

Usage:          makedisplay <sc> <im1> -motion <map> <im2> <dispout>

Colour map:     Image 0-255
--------------------------------------------------------------- */
        if(strcmp(argv[3],"-motion") == 0)
        {
                pixel *temp1,*temp2,*temp3,*temp4,*temp5;
                struct vectormap *matchpointer;
                struct vectormapheader vectmapheader;
```

```
                int x1,y1,z1,x2,y2,z2,pos1,pos2;

                temp1=imagetomem(argv[2],&width,&height,&frames);
                temp3=imagetomem(argv[5],&width,&height,&frames);
                temp2 = (pixel *)malloc(width*height*sizeof(pixel));
                temp4 = (pixel *)malloc(2*width*height*sizeof(pixel));
                temp5 = (pixel *)malloc(3*width*height*sizeof(pixel));
                picture = (pixel *)malloc(3*width*height*scale*scale*
                                                        sizeof(pixel));

                matchpointer=vectormaptomem(argv[4],&vectmapheader);
                for(count=0;count<vectmapheader.vectorcount;count++)
                {
                        x1 = (matchpointer[count].mapping).beg[0];
                        y1 = (matchpointer[count].mapping).beg[1];
                        z1 = (matchpointer[count].mapping).beg[2];
                        x2 = (matchpointer[count].mapping).end[0];
                        y2 = (matchpointer[count].mapping).end[1];
                        z2 = (matchpointer[count].mapping).end[2];
                        pos1=x1+y1*width+z1*width*height;
                        pos2=x2+y2*width+z2*width*height;
                        temp2[pos2]=temp1[pos1];
                }
                concatimage(temp1,temp2,temp4,width,width,height);
                free(temp1);
                free(temp2);
                concatimage(temp4,temp3,temp5,2*width,width,height);
                free(temp3);
                free(temp4);
                enlarge(temp5,picture,3*width,height,scale,"one");
                rastwidth = 3*width*scale;
                rastheight = height*scale;
                free(temp5);
                imagetofile(argv[6],picture,rastwidth,rastheight,frames);
        }
/* ---------------------------------------------------------------
Option:         -motdisp

Usage:          makedisplay <sc> <im1> -motdisp <map> <dispout>

Colour map:     Image 0-255
--------------------------------------------------------------- */
        if(strcmp(argv[3],"-motdisp") == 0)
        {
                pixel *temp1,*temp2;
                struct vectormap *matchpointer;
                struct vectormapheader vectmapheader;
                int x1,y1,z1,x2,y2,z2,pos1,pos2;

                temp1=imagetomem(argv[2],&width,&height,&frames);
                temp2 = (pixel *)malloc(width*height*sizeof(pixel));
                picture = (pixel *)malloc(width*height*scale*scale*
```

```
                                                        sizeof(pixel));

                matchpointer=vectormaptomem(argv[4],&vectmapheader);
                for(count=0;count<vectmapheader.vectorcount;count++)
                {
                        x1 = (matchpointer[count].mapping).beg[0];
                        y1 = (matchpointer[count].mapping).beg[1];
                        z1 = (matchpointer[count].mapping).beg[2];
                        x2 = (matchpointer[count].mapping).end[0];
                        y2 = (matchpointer[count].mapping).end[1];
                        z2 = (matchpointer[count].mapping).end[2];
                        pos1=x1+y1*width+z1*width*height;
                        pos2=x2+y2*width+z2*width*height;
                        temp2[pos2]=temp1[pos1];
                }
                free(temp1);
                enlarge(temp2,picture,width,height,scale,"one");
                rastwidth = width*scale;
                rastheight = height*scale;
                imagetofile(argv[5],picture,rastwidth,rastheight,frames);
        }
/* ---------------------------------------------------------------
Option:         -concat

Usage:          makedisplay <sc> <im1> -concat <im2> dispout

Comments:       The two images must be the same height.

Colour map:     Image 0-255.
--------------------------------------------------------------- */
        if(strcmp(argv[3],"-concat") == 0)
        {
                pixel *temp1,*temp2,*temp3;
                temp1=imagetomem(argv[2],&width,&height,&frames);
                temp2=imagetomem(argv[4],&width,&height,&frames);
                temp3 = (pixel *)malloc(2*width*height*sizeof(pixel));
                picture = (pixel *)malloc(2*width*height*scale*scale*
                                                        sizeof(pixel));
                concatimage(temp1,temp2,temp3,width,width,height);
                enlarge(temp3,picture,2*width,height,scale,"one");
                rastwidth = 2*width*scale;
                rastheight = height*scale;
                free(temp1);
                free(temp2);
                free(temp3);
                imagetofile(argv[5],picture,rastwidth,rastheight,frames);
        }
/* ---------------------------------------------------------------
Option:         No option chosen.

Usage:          makedisplay <sc> <im1> <dispout>
```

```
Comments:       Used for enlarging images.

Colour map:     Image 0-255.
------------------------------------------------------------------ */
        if(argc == 4)
        {
                pixel *temp;
                temp = imagetomem(argv[2],&width,&height,&frames);
                picture = (pixel *)malloc(width*height*scale*scale*
                                                        sizeof(pixel));
                enlarge(temp,picture,width,height,scale,"");
                rastwidth = width*scale;
                rastheight = height*scale;
                free(temp);
                imagetofile(argv[3],picture,rastwidth,rastheight,frames);
        }

}
```

B.3.13 masksio.c

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

Function:       regiontomem

Arguments:      filename - The name of a file containing the text
                        definition of a smoothing region.

Returns:        A pointer to a structure defining the region.

Dependancies:  fileio.c - ImageDefs.h

Side FX:        Allocates the required memory space.
------------------------------------------------------------------ */

struct region *regiontomem(filename)
char *filename;
{
        FILE *filepntr;
        int size,count;
        struct region *regpntr;

        filepntr = fopencheck(filename,"r","REGION");

        fscanf(filepntr,"%d\n",&size);

        regpntr =(struct region *)malloc(sizeof(int)+sizeof(struct offset
*));
        regpntr->points = (struct offset *)malloc(size*sizeof(struct offset));
```

```
        regpntr->regionsize=size;

        for(count=0;count<size;count++)
        {
                offsettomem(filepntr,&(regpntr->points[count]));
        }

        fclosecheck(filepntr,"REGION");
        return(regpntr);
}

/* ------------------------------------------------------------------
Function:       offsettomem(filepntr,offsetpntr)

Arguments:      filepntr - Pointer to an opened file
                offsetpntr - Pointer to memory allocated for offset struct

Returns:        VOID

Side FX:        Loads the value of the offset from the file into the
                memory allocated in the function regiontomem
------------------------------------------------------------------ */

offsettomem(filepntr,offsetpntr)
FILE *filepntr;
struct offset *offsetpntr;
{
        int *px,*py,*pz;

        px = &(offsetpntr->value[0]);
        py = &(offsetpntr->value[1]);
        pz = &(offsetpntr->value[2]);

        fscanf(filepntr,"%d,%d,%d\n",px,py,pz);
}
/* ------------------------------------------------------------------
Function:       masktomem(filename)

Arguments:      filename - The name of a file containing a mask.

Returns:        A pointer to the loaded mask structure.

Side FX:        Assigns memory for the mask.
------------------------------------------------------------------ */

struct local_mask masktomem(filename)
char *filename;
{
        int masksize,count;
        FILE *filepointer;
        struct local_mask mask;
```

```
        filepointer=fopencheck(filename,"r","MASK");

        fscanf(filepointer,"%d\n",&masksize);

        mask.mask_size=masksize;
        mask.mask_overlay=(int *)malloc(masksize*sizeof(int));

        for(count=0;count<masksize;count++)
        {
                fscanf(filepointer,"%d\n",&mask.mask_overlay[count]);
        }
        fclosecheck(filepointer,"MASK");
        return(mask);
}
```

B.3.14 remove.c

```
/* ------------------------------------------------------------------
Program:        remove

Function:       Removes the effects of image transformation from a
                vector  mapping.

Usage:          remove <vectormappingfile> mx my tx ty <op/file>

Functions:      NONE

Dependancies:   vectorio.c, fileio.c - ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */

#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char *argv[];
{
        struct vectormapheader vmaphdr;
        struct vectormap *vmap;
        int count;
        double mx,my,tx,ty;

        vmap=vectormaptomem(argv[1],&vmaphdr);

        sscanf(argv[2],"%lf",&mx);
        sscanf(argv[3],"%lf",&my);
        sscanf(argv[4],"%lf",&tx);
        sscanf(argv[5],"%lf",&ty);
```

```
        printf("Params %f %f %f %f\n",mx,my,tx,ty);

        for(count=0;count<vmaphdr.vectorcount;count++)
        {
                double xnew,ynew,xold,yold,xmap,ymap;

                xold=vmap[count].mapping.beg[0];
                yold=vmap[count].mapping.beg[1];
                xmap=vmap[count].mapping.end[0];
                ymap=vmap[count].mapping.end[1];

                xnew=mx*xold-my*yold+tx;
                ynew=mx*yold+my*xold+ty;

                vmap[count].mapping.end[0]=xold+xmap-xnew;
                vmap[count].mapping.end[1]=yold+ymap-ynew;
        }

        vectormaptofile(argv[6],vmap,vmaphdr);
}
```

B.3.15 smoothmap.c

```
/* ----------------------------------------------------------------
Program:        smoothmap

Function:       Smooths the initial mapping.

Usage:          smoothmap <gain> <iteration> <initialmap>
                                              <smoothed_out>

Functions:      struct vectormap smooth()
                struct offset arrowplus()

Dependancies:  vectorio.c, vectorfns.c, conversions.c, fileio.c,
                masksio.c - ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
---------------------------------------------------------------- */

#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char *argv[];
{
        int count,min[3],max[3],vcount,sampling;
        int axis,itercount,iterations;
```

```
        double alpha1,alpha2;
        struct vectormapheader vmaphdr;
        struct vectormap *vmap,*newmap,*oldmap;
        struct region *rmap;

        vmap = vectormaptomem(argv[1],&vmaphdr);
        sscanf(argv[2],"%lf",&alpha2);
        sscanf(argv[3],"%d",&iterations);

alpha1=1;
        printf("Aplha2 = %f\n",alpha2);

        vcount=vmaphdr.vectorcount;
        sampling=vmaphdr.sampling;

        newmap=(struct vectormap *)malloc(vcount*sizeof(struct vectormap));

        for(axis=0;axis<3;axis++)
        {
                min[axis]=vmaphdr.area.beg[axis];
                max[axis]=vmaphdr.area.end[axis];
                printf("%d min, %d max, %d axis\n",min[axis],max[axis],axis);
        }

        rmap = regiontomem(argv[4]);

        oldmap = vmap;

        printf("No of iterations to complete, %d\n",iterations);
        printf("No of vectors to process, %d\n",vcount);


        for(itercount=0;itercount<iterations;itercount++)
        {
           printf("Iteration %d\n",itercount+1);
           fflush(stdout);
           for(count=0;count<vcount;count++)
           {
                newmap[count] = smooth(vmap,oldmap,rmap,count,min,max,
                                                sampling,alpha1,alpha2);
           }
                oldmap=newmap;
        }
        vectormaptofile(argv[5],newmap,vmaphdr);
}

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

Function:       smooth

Arguments:  vmap - A vector mapping.
                oldmap - the previous iteration.
```

```
                rmap - The neighbourhood.
count - The position of the pixel in the array.
min - A pointer to an array containing the minimum x, y
and z of the area mapped.
max - A pointer to an array containing the maximum x, y
and z of the area mapped.
sampling - Sampling rate.
alpha1 - Always 1.
alpha2 - The neighbourhood gain.

Returns:  A pointer to a new vector mapping. The smoothed vector.
------------------------------------------------------------------- */

struct vectormap smooth(vmap,oldmap,rmap,count,min,max,sampling,
                                                    alpha1,alpha2)
struct vectormap *vmap,*oldmap;
struct region *rmap;
int min[3],max[3],sampling,count;
double alpha1,alpha2;
{
        struct vectormap newmap;
        double topval[3],bottomval[3];
        int axis,regcount;

        newmap.mapping.beg[0]=oldmap[count].mapping.beg[0];
        newmap.mapping.beg[1]=oldmap[count].mapping.beg[1];
        newmap.mapping.beg[2]=oldmap[count].mapping.beg[2];
        newmap.strength=oldmap[count].strength;

        for(axis=0;axis<3;axis++)
        {
                topval[axis]=alpha1*vmap[count].mapping.end[axis]*
                                            vmap[count].strength;
                bottomval[axis]=alpha1*vmap[count].strength;
        }

        for(regcount=0;regcount<rmap->regionsize;regcount++)
        {
                struct offset pos,newarrow;
                int newcount;

                pos=linetothreed(count,min,max,sampling);
                newarrow= arrowplus(pos,rmap->points[regcount],sampling);

                if(!outofrange(newarrow,min,max))
                {
                   newcount=threedtoline(newarrow,min,max,sampling);

                   for(axis=0;axis<3;axis++)
                   {
                      topval[axis]=topval[axis]+alpha2*
                       oldmap[newcount].mapping.end[axis]*
```

```
                                vmap[newcount].strength;
                        bottomval[axis]=bottomval[axis]+alpha2*
                                        vmap[newcount].strength;
                    }


                }

        }

        for(axis=0;axis<3;axis++)
        {
                if(bottomval[axis]==0)
                        newmap.mapping.end[axis]=0;
                else
                newmap.mapping.end[axis]=(topval[axis]/bottomval[axis]);
        }
        return(newmap);
}

struct offset arrowplus(a1,a2,sampling)
struct offset a1,a2;
int sampling;
{
        int axis;
        struct offset sum;

        for(axis=0;axis<3;axis++)
        {
                sum.value[axis]=a1.value[axis]+a2.value[axis]*sampling;
        }

        return(sum);
}
```

B.3.16 spotnoise.c

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

Program:        spotnoise

Function:       Performs conservative smoothing over an image to remove
                spot noise.

Usage:          spotnoise <image> <smoothed image> <region mask>

Dependancies:   imagio.c, imfunctions.c, fileio.c - ImageDefs.h

Written:        Nick Woodward MSc 1989-1990
```

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

#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char **argv;
{
        pixel *imagein,*imageout;
        struct local_mask mask;
        int width,height,frames;

        imagein=imagetomem(argv[1],&width,&height,&frames);

        imageout = (pixel *)malloc(width*height*frames*sizeof(pixel));

        mask = masktomem(argv[3]);

        remspotnoise(imagein,imageout,mask,width,height);

        imagetofile(argv[2],imageout,width,height,frames);

}
```

B.3.17 stretch.c

```
/* ------------------------------------------------------------------
Program:        stretch

Function:       Normalises the weights associated with a vector mapping
                so that the range lies between 0 and 1.

Usage:          stretch <inputfile> <outputfile>

Dependancies:   fileio.c, vectorio.c - ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */

#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char ** argv;
{
        struct vectormap *coordmap;
        struct vectormapheader coordmaphdr;
        int count;
```

```
        double max,min;

        max = 0;
        min = 1;

        coordmap = vectormaptomem(argv[1],&coordmaphdr);

        fprintf(stdout,"Stretching starting\n");

        for(count=0;count<(coordmaphdr.vectorcount);count++)
        {
                if(coordmap[count].strength<min)
                        min=coordmap[count].strength;
                if(coordmap[count].strength>max)
                        max=coordmap[count].strength;
        }

        for(count=0;count<(coordmaphdr.vectorcount);count++)
        {
                coordmap[count].strength=(coordmap[count].strength-min)
                                                                /(max-min);
        }

        fprintf(stdout,"Stretching complete\n");

        vectormaptofile(argv[2],coordmap,coordmaphdr);
}
```

B.3.18 transform.c

```
/* ------------------------------------------------------------------
Program:        transform

Function:       Calculates the transformation parameters between two
                sets of points. The parameters are output to stdout.

Usage:          transform <vectormapfile>

Functions:      ludcmp()
                lubksb()
                nrerror()
                free_vector()
                double *vector
                double **matrix(rows,cols)
                double **transpose(matx,cols,rows)
                double **matrixmultiply()
                display matrix()
                double **invertmatrix()

Dependencies:   vectorio.c, fileio.c - ImageDefs.h
```

```
Comments:       This major part of this code is taken from Numerical
                Recipes  in 'C'.
                The transformation calculates a least squares fit
                of two sets of image points to estimate the rotation,
                magnification and translation between the two sets
                of points. See Meertens, 'Field Placement Errors in
                Digital Portal Images', Physics in Medicine and Biology,
                Vol. 35, No 3, p318, 1990.

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------- */

#include <stdio.h>
#include "ImageDefs.h"

#define TINY 1.0e-20;

double **matrix();
double **transpose();
double **matrixmultiply();
double **invertmatrix();

void main(argc,argv)
int argc;
char *argv[];
{
        struct vectormap *vmap;
        struct vectormapheader vmaphdr;
        double **amatrix,**pmatrix,**mmatrix,**atmatrix,**atamatrix;
        double **invmatrix,**test1,**test2,**ident;
        int points,count,matrixcount,i,j;
        double scale,rot,m1,m2,tx,ty,arctval;

        vmap = vectormaptomem(argv[1],&vmaphdr);

        points=vmaphdr.vectorcount;

        amatrix=matrix(4,2*points);
        pmatrix=matrix(1,2*points);
        test1=matrix(4,4);

        matrixcount=1;
        for(count=1;count<=points;count++)
        {
                pmatrix[1][matrixcount++] = vmap[count-1].mapping.end[0];
                pmatrix[1][matrixcount++] = vmap[count-1].mapping.end[1];
        }

        matrixcount=1;
        for(count=1;count<=points;count++)
        {
```

```
                amatrix[1][matrixcount]=vmap[count-1].mapping.beg[0];
                amatrix[2][matrixcount]=vmap[count-1].mapping.beg[1] *-1;
                amatrix[3][matrixcount]=1;
                amatrix[4][matrixcount++]=0;
                amatrix[1][matrixcount]=vmap[count-1].mapping.beg[1];
                amatrix[2][matrixcount]=vmap[count-1].mapping.beg[0];
                amatrix[3][matrixcount]=0;
                amatrix[4][matrixcount++]=1;
        }

        printf("Completed assignment\n");

        atmatrix=transpose(amatrix,4,2*points);

        atamatrix=matrixmultiply(atmatrix,2*points,4,amatrix,4,2*points);
        for(i=1;i<=4;i++)
        for(j=1;j<=4;j++)
                test1[i][j]=atamatrix[i][j];

        invmatrix=invertmatrix(atamatrix,4);
        ident=matrixmultiply(test1,4,4,invmatrix,4,4);
        printf("-----------------------------------------\n");
        displaymatrix(ident,4,4);

        test2=matrixmultiply(invmatrix,4,4,atmatrix,2*points,4);
        mmatrix=matrixmultiply(test2,2*points,4,pmatrix,1,2*points);

        m1=mmatrix[1][1];
        m2=mmatrix[1][2];
        printf("Params = %f,%f\n",m1,m2);
        tx=mmatrix[1][3];
        ty=mmatrix[1][4];
        arctval=m2/m1;
        printf("Arctanvalue = %f\n",arctval);
        rot=atan((double)arctval)*360.0/2/3.14159;
        printf("Translation is (%f,%f)\n",tx,ty);
        scale=sqrt(m1*m1+m2*m2);
        printf("Scale is %f\n",scale);
        printf("Rotation is %f\n",rot);
}

void ludcmp(a,n,indx,d)
int n, *indx;
double **a,*d;
{
        int i,imax,j,k;
        double big,dum,sum,temp;
        double *vv,*vector();
        void nrerror(),free_vector();


        vv=vector(1,n);
```

```
        *d=1.0;

        for(i=1;i<=n;i++)
        {
                big=0.0;
                for(j=1;j<=n;j++)
                        if((temp=fabs(a[i][j]))>big) big=temp;
                if(big==0.0) nrerror("Singular matrix in routine ludcmp");
                vv[i]=1.0/big;
        }
        for(j=1;j<=n;j++)
        {
                for(i=1;i<j;i++)
                {
                        sum=a[i][j];
                        for(k=1;k<i;k++) sum -=a[i][k]*a[k][j];
                        a[i][j]=sum;
                }
                big=0.0;
                for(i=j;i<=n;i++)
                {
                        sum=a[i][j];
                        for(k=1;k<j;k++) sum -=a[i][k]*a[k][j];
                        a[i][j]=sum;
                        if((dum=vv[i]*fabs(sum)) >=big)
                        {
                                big=dum;
                                imax=i;
                        }
                }
                if(j!=imax)
                {
                        for(k=1;k<=n;k++)
                        {
                                dum=a[imax][k];
                                a[imax][k]=a[j][k];
                                a[j][k]=dum;
                        }
                        *d= -(*d);
                        vv[imax]=vv[j];
                }
                indx[j]=imax;
                if(a[j][j]==0.0) a[i][j]=TINY;
                if(j!=n)
                {
                        dum=1.0/(a[j][j]);
                        for(i=j+1;i<=n;i++) a[i][j] *=dum;
                }
        }       free_vector(vv,1,n);
}

void lubksb(a,n,indx,b)
```

```
double **a,b[];
int n,*indx;
{
        int i,ii=0,ip,j;
        double sum;

        for(i=1;i<=n;i++)
        {
                ip=indx[i];
                sum=b[ip];

                b[ip]=b[i];
                if(ii)
                        for(j=ii;j<=i-1;j++)
                                sum -=a[i][j]*b[j];
                else if(sum) ii=i;
                b[i]=sum;
        }
        for(i=n;i>=1;i--)
        {
                sum=b[i];
                for(j=i+1;j<=n;j++) sum -=a[i][j]*b[j];
                b[i]=sum/a[i][i];
        }
}

void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr,error_text);

        exit(0);
}

void free_vector(v,nl,nh)
double *v;
int nl,nh;
{
        free((char*)(v+nl));
}

double *vector(nl,nh)
int nl,nh;
{
        double *v;

        v = (double*)malloc ((unsigned)(nh-nl+1)*sizeof(double));
        if(!v) nrerror("Allocation failure in vector()");
        return v-nl;
}
```

```
double **matrix(rows,cols)
int rows,cols;
{
        double **matx;
        int count;
        matx=(double **)malloc((rows+1)*sizeof(double *));
        for(count=1;count<=rows;count++)
        {
                matx[count]=(double *)malloc((cols+1)*sizeof(double));
        }
        return(matx);
}

double **transpose(matx,cols,rows)
double **matx;
int cols,rows;
{
        double **trans;
        int i,j;

        trans=matrix(rows,cols);

        printf("Transposing\n");

        for(i=1;i<=cols;i++)
        {
                for(j=1;j<=rows;j++)
                {
                        trans[j][i]=matx[i][j];
                }
        }

        printf("Transposed\n");
        return(trans);
}

double **matrixmultiply(m1,col1,row1,m2,col2,row2)
double **m1,**m2;
int row1,row2,col1,col2;
{
        double **matx,sum;
        int i,j,k;

        if(col1!=row2) rterror("Matrices incompatible : Fn -> matrix()");
        matx=matrix(col2,row1);
        if(matx == NULL)
        {
                rterror("Memory error : Fn -> matrix()");
        }

        printf("Starting multiplication\n");
```

```
        for(k=1;k<=col2;k++)
        {
                for(i=1;i<=row1;i++)
                {
                        sum=0.0;
                        for(j=1;j<=col1;j++)
                        {
                                sum +=m1[j][i]*m2[k][j];
                        }
                        matx[k][i]=sum;
                }
        }
        return(matx);
}

displaymatrix(matx,cols,rows)
double **matx;
int rows,cols;
{
        int i,j;

        for(i=1;i<=rows;i++)
        {
                printf("|");
                for(j=1;j<=cols;j++)
                {
                        printf("%lf ",matx[j][i]);
                }
                printf("|\n");
        }
}

rterror(errm)
char *errm;
{
        fprintf(stderr,"Runtime error\n");
        fprintf(stderr,errm);
        fprintf(stderr,"\n");
        exit(0);
}

double **invertmatrix(a,N)
double **a;
{
        int i,j,*indx;
        double d,*col,**y;

        y=matrix(N,N);
        indx=(int *)malloc((N+1)*sizeof(int));

        col=(double *)malloc((N+1)*sizeof(double));
```

```
        ludcmp(a,N,indx,&d);
        for(j=1;j<=N;j++)
        {
                for(i=1;i<=N;i++) col[i]=0.0;
                col[j]=1.0;
                lubksb(a,N,indx,col);
                for(i=1;i<=N;i++) y[i][j]=col[i];
        }
        return(y);
}
```

B.3.19 transthresh.c

```
/* ------------------------------------------------------------------
Program:        transthresh

Function:       Threshold an image and removes vector mappings where the
                raw data value is below a threshold. Intended to leave
                only mappings from relevant data pixels before the image
                transformation is calculated.

Usage:          transthresh <ip/mapping> <op/mapping> <threshold>

Dependancies:   vectorio.c, imageio.c, fileio.c

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */

#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char *argv[];
{
        int threshold,count,width,height,frames,cnt;
        pixel *imagein;
        struct vectormapheader vmaphdr;
        struct vectormap *vmap;

        imagein=imagetomem(argv[1],&width,&height,&frames);

        sscanf(argv[3],"%d",&threshold);

        vmap=vectormaptomem(argv[2],&vmaphdr);

        cnt=0;

        for(count=0;count<vmaphdr.vectorcount;count++)
```

```
        {
                int posn;
                double x,y;

                x=vmap[count].mapping.beg[0];
                y=vmap[count].mapping.beg[1];

                posn=(int)x+(int)y*width;

                if((unsigned int)imagein[posn]<threshold);
                else
                        vmap[cnt++]=vmap[count];
        }
        vmaphdr.vectorcount=cnt;

        vectormaptofile(argv[4],vmap,vmaphdr);
}
```

B.3.20 vectmap.c

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

Program:        vectmap

Function:       Takes two image files and correlates a set of points in
                the first image to the best point that lies within a
                particular region around that point in the other image.
                The output is a list of which points are matched to
                which points, listed on the screen.

Usage:          vectmap <imagefile1> <imagefile2> <mask> <region> <width>
                                <height> <sampling> <O/Pfile>

Comments:       The sampling rate determines how many pixels in an image
                are found matches. This was only of use when developing
                the mapping and should be 1 normally.

Dependancies:   imageio.c, vectorio.c, vectorfns.c, masksio.c, fileio.c,
                conversions.c - ImageDefs.h

Written:        Nick Woodward (MSc 1989-90) 5th June 1990.

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

#include <stdio.h>
#include "ImageDefs.h"

main(argc,argv)
int argc;
char **argv;
```

```
{
        pixel *image1,*image2;
        struct local_mask corrmask,corrregion;
        struct vectormap *matches;
        struct vectormapheader vectheader;
        int *postomatch,regcount,bestpos,newpos,count,regsize;
        int x1,x2,y1,y2,width,height,pos,sampling,samplecount,frames;
        double maxcorr,corr,xbeg,ybeg,zbeg,xfin,yfin,zfin;
        int min[3],max[3],axis;

        bestpos = 0;

        fprintf(stdout,"Correlation testing\n");
        sscanf(argv[5],"%d",&sampling);

        fprintf(stdout,"Input area delimiters\n");
        scanf("%d,%d,%d,%d\n",&min[0],&min[1],&max[0],&max[1]);
        fprintf(stdout,"Area defined by - ");
        fprintf(stdout,"%d %d %d %d\n",min[0],min[1],max[0],max[1]);

        min[2]=max[2]=0;

        samplecount=1;
        for(axis=0;axis<3;axis++)
        {
                int axislength;
                axislength=max[axis]/sampling-min[axis]/sampling;
                if(min[axis]%sampling==0) axislength++;
                samplecount=samplecount*axislength;
        }
        printf("Sample size %d\n",samplecount);

        postomatch=(int *)malloc(samplecount*sizeof(int));
        matches=(struct vectormap *)malloc(samplecount*
                                sizeof(struct vectormap));

        image1=imagetomem(argv[1],&width,&height,&frames);
        image2=imagetomem(argv[2],&width,&height,&frames);

        corrmask=masktomem(argv[3]);
        corrregion=masktomem(argv[4]);

        printf("Both masks loaded\n");

        pos = 0;

        for(count=0;count<width*height;count++)
        {
                struct offset coord;

                coord=counttoxyz(count,width,height);
```

```
        if(!outofrange(coord,min,max))
        {
                if(coord.value[0]%sampling>0) continue;
                if(coord.value[1]%sampling>0) continue;
                if(coord.value[2]%sampling>0) continue;
                postomatch[pos++]= count;
        }
}

printf("Samplecount = %d ,(%d)\n",samplecount,pos);

regsize = corrregion.mask_size;

for(count=0;count<pos;count++)
{
        struct vector vtemp;
        double flag;

        if (count%100 == 0)
        {
                fprintf(stdout,"Count %d\n",count);
                fflush(stdout);
        }

        flag = 1.0;
        maxcorr = 0;
        for(regcount=0;regcount<regsize;regcount++)
        {
                struct offset coord1,coord2;
                int minborder[3],maxborder[3];

                minborder[0]=minborder[1]=minborder[2];
                maxborder[0]=width-1;
                maxborder[1]=height-1;
                maxborder[2]=0;

                newpos=postomatch[count]+
        corrregion.mask_overlay[regcount];
                coord1=counttoxyz(newpos,width,height);
                coord2=counttoxyz(postomatch[count],width,height);

                if(coord1.value[0] < 22 && coord2.value[0] > 100)
                        continue;
                if(coord2.value[0] < 22 && coord1.value[0] > 100)
                        continue;

                if(!outofrange(coord1,minborder,maxborder))
                {

                corr =(double)correlate(postomatch[count],image1,
                           newpos,image2,corrmask,width,height);
                if(newpos < 0 || bestpos > width*height-1)
```

```
                {
                        printf("Out of range error (Best Match)\n");
                        printf("Count = %d\n",count);
                        printf("Mask value = %d\n",
                                corrregion.mask_overlay[regcount]);
                        printf("Position = %d\n",postomatch[count]);
                        exit(0);
                }
                if (corr == maxcorr)
                {
                        flag += 1.0;
                }
                if (corr > maxcorr)
                {
                        bestpos=newpos;
                        maxcorr = corr;
                        flag = 1.0;
                }

                }

        }
        if(maxcorr == 0)
        {
                bestpos = postomatch[count];
        }
        x1 = (postomatch[count]%width);
        y1 = (postomatch[count]/width);
        x2 = (bestpos%width);
        y2 = (bestpos/width);

        vtemp.beg[0]=x1;
        vtemp.beg[1]=y1;
        vtemp.beg[2]=0;
        vtemp.end[0]=x2;
        vtemp.end[1]=y2;
        vtemp.end[2]=0;

        matches[count].mapping=vtemp;
        matches[count].strength= maxcorr/flag;
}

vectheader.image_dims[0] = width;
vectheader.image_dims[1] = height;
vectheader.image_dims[2] = 1;
strcpy(vectheader.imagefrom,argv[1]);
strcpy(vectheader.imageto,argv[2]);
vectheader.vectorcount = samplecount;
vectheader.sampling = sampling;
strcpy(vectheader.corrmask,argv[3]);
strcpy(vectheader.regionmask,argv[4]);
```

```
        xbeg = matches[0].mapping.beg[0];
        ybeg = matches[0].mapping.beg[1];
        zbeg = matches[0].mapping.beg[2];
        xfin = matches[samplecount-1].mapping.beg[0];
        yfin = matches[samplecount-1].mapping.beg[1];
        zfin = matches[samplecount-1].mapping.beg[2];

        setvector(&(vectheader.area),xbeg,ybeg,zbeg,xfin,yfin,zfin);

        vectormaptofile(argv[6],matches,vectheader,"w");

        free(postomatch);
        free(image1);
        free(image2);
        free(corrregion.mask_overlay);
        free(matches);

        return 0;
}
```

B.3.21 vectorfns.c

```
/* ------------------------------------------------------------------
Program:        vectorfns.c

Function:       Functions to manipulate vectors

Usage:          Called from othercode only.

functions:      setvector()
                displayvector()
                Both are self explanantory.

Written:        Nick Woodward, MSc 1989-1990
------------------------------------------------------------------ */

#include <stdio.h>
#include "ImageDefs.h"

setvector(vectpntr,x1,y1,z1,x2,y2,z2)
struct vector *vectpntr;
double x1,y1,z1,x2,y2,z2;
{
        vectpntr->beg[0] = x1;
        vectpntr->beg[1] = y1;
        vectpntr->beg[2] = z1;
        vectpntr->end[0] = x2;
        vectpntr->end[1] = y2;
        vectpntr->end[2] = z2;
}
```

```
displayvector(vect)
struct vector vect;
{
        printf("From %f,%f,%f",vect.beg[0],vect.beg[1],vect.beg[2]);
        printf(" to %f,%f,%f\n",vect.end[0],vect.end[1],vect.end[2]);
}
```

B.3.22 vectorio.c

```
/* ---------------------------------------------------------------
Program:        vectorio (Not executable)

Function:       Provides functions to save,
                read, and display information about
                the vector mapping between two images.

Usage:          Called from other code only.

functions:      vectormaptofile()
                vectorheadertofile()
                mappingtofile()
                struct vectormap *vectormaptomem()
                vectorheadertomem()
                mappingtomem()
                struct vectormap *relaxmaptomem()
                displayvectormapheader()

Dependancies:   fileio.c - ImageDefs.h

Written:        Nick Woodward, MSc 1989-1990
--------------------------------------------------------------- */

#include <stdio.h>
#include "ImageDefs.h"

/* ---------------------------------------------------------------
Function:       vectormaptofile

Arguments:      filename - File to write.
                vectormappntr - A pointer to the start of
                        the vector mapping.
                vectheader - A structure containing information
                        about the mapping.

Returns:        VOID

Side FX:        Generates a text file containing a header and vector
                        mappings.
```

```
Comments:       See Headers.h for a definition of the vectorheader structure.
                See ImageDefs.h for a definition of the vectormap structure.
---------------------------------------------------------------- */

vectormaptofile(filename,vectormappntr,vectheader)
char *filename;
struct vectormap *vectormappntr;
struct vectormapheader vectheader;
{
        int count;
        FILE *filepntr;

        filepntr = fopencheck(filename,"w","VECTORMAP");

        vectorheadertofile(filepntr,vectheader);

        fprintf(stdout,"Writing vector map - ");
        for(count=0;count < vectheader.vectorcount;count++)
        {
                mappingtofile(filepntr,&(vectormappntr[count]));
        }
        fprintf(stdout,"Written\n");
        fclosecheck(filepntr,"VECTORMAP");
}

/* ----------------------------------------------------------------
Function:       vectorheadertofile

Arguments:      filepntr - Pointer to an opened file into which
                        to write the header information.
                vectheader - A vectormapheader structure containing
                        information on the generation of the vector
                        map.

Returns:        VOID

Side FX:        Writes text to the file.

Comments:       See Headers.h for a definition of the vectorheader structure.
                See ImageDefs.h for a definition of the vectormap structure.
---------------------------------------------------------------- */

vectorheadertofile(filepntr,vectheader)
FILE *filepntr;
struct vectormapheader vectheader;
{
        int xdims,ydims,zdims;
        double x1,y1,z1,x2,y2,z2;

        fprintf(stdout,"Writing vector map header - ");

        xdims=vectheader.image_dims[0];
```

```
        ydims=vectheader.image_dims[1];
        zdims=vectheader.image_dims[2];
        x1 = vectheader.area.beg[0];
        y1 = vectheader.area.beg[1];
        z1 = vectheader.area.beg[2];
        x2 = vectheader.area.end[0];
        y2 = vectheader.area.end[1];
        z2 = vectheader.area.end[2];

        fprintf(filepntr,"Image Dimensions, %d (x) by %d (y) by %d
                                                (z)\n",xdims,ydims,zdims);
        fprintf(filepntr,"Image %s mapped ",vectheader.imagefrom);
        fprintf(filepntr,"to image %s\n",vectheader.imageto);
        fprintf(filepntr,"Vector count = %d\n",vectheader.vectorcount);
        fprintf(filepntr,"Sampling rate, %d\n",vectheader.sampling);
        fprintf(filepntr,"Region mask, %s\n",vectheader.regionmask);
        fprintf(filepntr,"Correlation mask, %s\n",vectheader.corrmask);
        fprintf(filepntr,"Start of area mapped, (%lf,%lf,%lf)\n",x1,y1,z1);
        fprintf(filepntr,"Finish of area mapped, (%lf,%lf,%lf)\n",x2,y2,z2);
        fflush(filepntr);

        fprintf(stdout,"Written\n");

}

/* ------------------------------------------------------------------
Function:       mappingtofile

Arguments:      filepntr - An opened file into which to write
                        the mapping data.
                mappingpntr - A pointer to a vectormap structure
                        which contains a vector mapping and its
                        strength.

Returns:        VOID

Side FX:        Writes text to the file.

Comments:       See Headers.h for a definition of the vectorheader structure.
                See ImageDefs.h for a definition of the vectormap structure.
------------------------------------------------------------------ */

mappingtofile(filepntr,mappingpntr)
FILE *filepntr;
struct vectormap *mappingpntr;
{
        double reliability;
        double x1,y1,z1,x2,y2,z2;

        x1 = (mappingpntr->mapping).beg[0];
        y1 = (mappingpntr->mapping).beg[1];
        z1 = (mappingpntr->mapping).beg[2];
```

```
                        which the mapping data is loaded.

Returns:        VOID

Side FX:        Reads in one piece of mapping data from the file.

Comments:       See ImageDefs.h for a definition of the vectormap structure.
---------------------------------------------------------------------- */

mappingtomem(filepntr,mappingpntr)
FILE *filepntr;
struct vectormap *mappingpntr;
{
        double reliability;
        double x1,y1,z1,x2,y2,z2;

        fscanf(filepntr,"%lf,%lf,%lf,%lf,%lf,%lf,%lf\n",&x1,&y1,&z1,&x2,
&y2,&z2,&reliability);

        (mappingpntr->mapping).beg[0] =x1;
        (mappingpntr->mapping).beg[1] =y1;
        (mappingpntr->mapping).beg[2] =z1;
        (mappingpntr->mapping).end[0] = x2;
        (mappingpntr->mapping).end[1] = y2;
        (mappingpntr->mapping).end[2] =z2;
        mappingpntr->strength = reliability;
}

/* ----------------------------------------------------------------------
Function:       displayvectormapheader

Arguments:      vectheader - A vectormapheaderstructure.

Returns:        VOID

Side FX:        Puts the information in the header to stdout.

Comments:       See Headers.h for a definition of the vectorheader structure.
---------------------------------------------------------------------- */

displayvectormapheader(vectheader)
struct vectormapheader vectheader;
{
        fprintf(stdout,"Image size: %d (x) by",vectheader.image_dims[0]);
        fprintf(stdout," %d (y) by",vectheader.image_dims[1]);
        fprintf(stdout," %d (z)\n",vectheader.image_dims[2]);
        fprintf(stdout,"Mapping from image, %s, to ",vectheader.imagefrom);
        fprintf(stdout,"image, %s.\n",vectheader.imageto);
        fprintf(stdout,"Using correlation mask %s\n",vectheader.corrmask);
        fprintf(stdout,"Over a region defined by %s\n",vectheader.regionmask);
        fprintf(stdout,"With a sampling rate of %d.\n",vectheader.sampling);
}
```