edge.c

Go to the documentation of this file.

```
/******************************************************************************/
/*                        Edge Detection Program                              */
/*  A suggested user interface is as follows:                                 */
/*  edge {-roberts,-prewitt,-sobel,-frei} [-skipNMS] [-t thresh1 thresh2] img > edgem*/
/*                    ECE 532 : Digital Image Analysis                        */
/*                              HW Assignment 1                               */
/*  Input: PGM file
    Output:PGM file + Image map.
    Author: Nikhil Shirahatti
    Date: 09/13/2003

*******************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

extern int read_pgm_hdr(FILE *, int *, int *);
extern void **matrix(int, int, int, int, int);
extern void error(const char *);
int skipNMS=0;


/*\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\*/

/* READ PGM HEADER */

/* This function reads the header of a PGM image. */
/* The dimensions are returned as arguments. */
/* This function ensures that there's no more than 8 bpp. */
/* The return value is negative if there's an error. */

int read_pgm_hdr(FILE *fp, int *nrows, int *ncols)
{
  char filetype[3];
  int maxval;

  if(skipcomment(fp) == EOF
     || fscanf(fp, "%2s", filetype) != 1
     || strcmp(filetype, "P5")
     || skipcomment(fp) == EOF
     || fscanf(fp, "%d", ncols) != 1
     || skipcomment(fp) == EOF
     || fscanf(fp, "%d", nrows) != 1
     || skipcomment(fp) == EOF
     || fscanf(fp, "%d%*c", &maxval) != 1
     || maxval > 255)
    return(-1);
  else return(0);
}

/*-------------------------------------------------------------------------------------*/
/* ERROR HANDLER */

void error(const char *msg)
{
  fprintf(stderr, "%s\n", msg);
  exit(1);
}

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

/* DYNAMICALLY ALLOCATE A PSEUDO 2-D ARRAY */

/* This function allocates a pseudo 2-D array of size nrows x ncols. */
/* The coordinates of the first pixel will be first_row_coord and */
/* first_col_coord. The data structure consists of one contiguous */
/* chunk of memory, consisting of a list of row pointers, followed */
/* by the array element values. */
/* Assumption:  nrows*ncols*element_size, rounded up to a multiple  */
/* of sizeof(long double), must fit in a long type.  If not, then */
/* the "i += ..." step could overflow.                              */

void **matrix(int nrows, int ncols, int first_row_coord,
              int first_col_coord, int element_size)
{
  void **p;
  int alignment;
  long i;

  if(nrows < 1 || ncols < 1) return(NULL);
  i = nrows*sizeof(void *);
  /* align the addr of the data to be a multiple of sizeof(long double) */
  alignment = i % sizeof(long double);
  if(alignment != 0) alignment = sizeof(long double) - alignment;
  i += nrows*ncols*element_size+alignment;
  if((p = (void **)malloc((size_t)i)) != NULL)
  {
    /* compute the address of matrix[first_row_coord][0] */
    p[0] = (char *)(p+nrows)+alignment-first_col_coord*element_size;
    for(i = 1; i < nrows; i++)
      /* compute the address of matrix[first_row_coord+i][0] */
      p[i] = (char *)(p[i-1])+ncols*element_size;
    /* compute the address of matrix[0][0] */
    p -= first_row_coord;
  }
  return(p);
}

/*-----------------------------------------------------------------------------------------------------------------*/
/* SKIP COMMENT */

/* This function skips past a comment in a file. The comment */
/* begins with a '#' character and ends with a newline character. */
/* The function returns EOF if there's an error. */

int skipcomment(FILE *fp)
```

```
00108 {
00109   int i;
00110 
00111   if((i = getc(fp)) == '#')
00112     while((i = getc(fp)) != '\n' && i != EOF);
00113   return(ungetc(i, fp));
00114 }
00115 
00116 
00117 /*-------------------------------------------------------------------------------------------------------------------------*/
00118 
00119 /* REFLECT AN IMAGE ACROSS ITS BORDERS */
00120 
00121 /* The parameter "amount" tells the number of rows or columns to be */
00122 /* reflected across each of the borders. */
00123 /* It is assumed that the data type is unsigned char. */
00124 /* It is assumed that the array was allocated to be of size at least */
00125 /* (nrows+2*amount) by (ncols+2*amount), and that the image was loaded */
00126 /* into the middle portion of the array, with coordinates, */
00127 /*      0 <= row < nrows, 0 <= col < ncols */
00128 /* thereby leaving empty elements along the borders outside the image */
00129 /* The "reflect" function will then fill in those empty */
00130 /* elements along the borders with the reflected image pixel values. */
00131 /* For example, x[0][-1] will be assigned the value of x[0][0], */
00132 /* and x[0][-2] will be assigned the value of x[0][1], if amount=2. */
00133 
00134 void reflect(unsigned char **xc, int nrows, int ncols, int amount)
00135 {
00136   int i, j;
00137 
00138   if(matrix == NULL || nrows < 1 || ncols < 1 || amount < 1
00139     || amount > (nrows+1)/2 || amount > (ncols+1)/2)
00140     error("reflect: bad args");
00141 
00142   for(i = -amount; i < 0; i++)
00143   {
00144     for(j = -amount; j < 0; j++)
00145       xc[i][j] = xc[-i-1][-j-1];
00146     for(j = 0; j < ncols; j++)
00147       xc[i][j] = xc[-i-1][j];
00148     for(j = ncols; j < ncols+amount; j++)
00149       xc[i][j] = xc[-i-1][ncols+ncols-j-1];
00150   }
00151   for(i = 0; i < nrows; i++)
00152   {
00153     for(j = -amount; j < 0; j++)
00154       xc[i][j] = xc[i][-j-1];
00155     for(j = ncols; j < ncols+amount; j++)
00156       xc[i][j] = xc[i][ncols+ncols-j-1];
00157   }
00158   for(i = nrows; i < nrows+amount; i++)
00159   {
00160     for(j = -amount; j < 0; j++)
00161       xc[i][j] = xc[nrows+nrows-i-1][-j-1];
00162     for(j = 0; j < ncols; j++)
00163       xc[i][j] = xc[nrows+nrows-i-1][j];
00164     for(j = ncols; j < ncols+amount; j++)
00165       xc[i][j] = xc[nrows+nrows-i-1][ncols+ncols-j-1];
00166   }
00167 }
00168 /*-------------------------------------------------------------------------------------------------------*/
00169 
00170 /* REFLECTING FLOAT */
00171 
00172 void reflectf(float **xc, int nrows, int ncols, int amount)
00173 {
00174  int i, j;
00175 
00176   if(matrix == NULL || nrows < 1 || ncols < 1 || amount < 1
00177     || amount > (nrows+1)/2 || amount > (ncols+1)/2)
00178     error("reflect: bad args");
00179 
00180 
00181   for(i = -amount; i < 0; i++)
00182   {
00183     for(j = -amount; j < 0; j++)
00184       xc[i][j] = xc[-i-1][-j-1];
00185     for(j = 0; j < ncols; j++)
00186       xc[i][j] = xc[-i-1][j];
00187     for(j = ncols; j < ncols+amount; j++)
00188       xc[i][j] = xc[-i-1][ncols+ncols-j-1];
00189   }
00190 
00191   for(i = 0; i < nrows; i++)
00192   {
00193     for(j = -amount; j < 0; j++)
00194       xc[i][j] = xc[i][-j-1];
00195     for(j = ncols; j < ncols+amount; j++)
00196       xc[i][j] = xc[i][ncols+ncols-j-1];
00197   }
00198 
00199   for(i = nrows; i < nrows+amount; i++)
00200   {
00201     for(j = -amount; j < 0; j++)
00202       xc[i][j] = xc[nrows+nrows-i-1][-j-1];
00203     for(j = 0; j < ncols; j++)
00204       xc[i][j] = xc[nrows+nrows-i-1][j];
00205     for(j = ncols; j < ncols+amount; j++)
00206       xc[i][j] = xc[nrows+nrows-i-1][ncols+ncols-j-1];
00207   }
00208 }
00209 /*-------------------------------------------------------------------------------------------------------------------------*/
00210 
00211 /* A method for Linear Interpolation */
00212 
00213 float LinearInterpolate(
00214    float y1,float y2,
00215    float mu)
00216 {
00217    return(y1*(1-mu)+y2*mu);
00218 }
00219 
00220 /*-------------------------------------------------------------------------------------------*/
00221 
00222 void nms(float **anglec, float **magc,int nrows , int ncols,int thresh,int thresh2)
00223 {
00224 
```

```
00225 /* PERFORMS NON-MAXIMUM SUPRESSION TO DETECT EDGES */
00226
00227   int i,j,r,c,edgepoints1=0,edgepoints2=0,last=0,nochange;
00228   FILE *fpy;
00229   float angle,maxm;
00230   int ax_pos,ay_pos,r1,c1,r2,c2,edgecount,r3,c3,r4,c4,count_points=0;
00231   unsigned char** edgemap;
00232   float mu =0.5 , anglecal, ivalue1,ivalue2;
00233   printf("============================================================\n");
00234   printf(" Performing Non_maximum-Supression\n");
00235
00236
00237   edgemap = (unsigned char **) matrix(nrows+2,ncols+2,-1,-1,sizeof(char));
00238   /* SO FIRST REFLECT THE MAGC BY 1 PIXEL */
00239   reflectf(magc, nrows, ncols, 1);
00240   reflectf(anglec,nrows,ncols,1);
00241   printf("------------------------------------------------\n");
00242   printf("Finished processing: Reflected magnitude image by 1 pixel\n");
00243   printf("Finished processing: Reflected angle image by 1 pixel\n");
00244
00245  /* HYSTERESIS THRESHODING */
00246
00247       for(i=0;i<nrows;i++)
00248         {
00249           for(j=0;j<ncols;j++)
00250             {
00251               edgemap[i][j] =0;
00252             }
00253         }
00254
00255       if(thresh != thresh2)
00256         {
00257           while( nochange)
00258             {
00259
00260               for(i=0;i<nrows;i++)
00261                 {
00262                   for(j=0;j<ncols;j++)
00263                     {
00264                       if(magc[i][j] >= thresh2 *thresh2)
00265                         {
00266                           edgemap[i][j] = 255;
00267                           edgepoints2++;
00268                         }
00269                       else if(magc[i][j] <= thresh*thresh)
00270                         {
00271                           edgemap[i][j] = 0;
00272                         }
00273                       else
00274                         {
00275                           edgecount =0;
00276                           for(r=i-1;r<= i+1; r++)
00277                             {
00278                               for(c=j-1;c<=j+1;c++)
00279                                 {
00280                                   if(edgemap[r][c] == 255 )
00281                                     edgecount++;
00282                                 }
00283                             }
00284                           if(edgecount >= 1)
00285                             {
00286                               edgemap[i][j] = 255;
00287                               edgepoints2++;
00288                             }
00289                           else
00290                             edgemap[i][j] =0;
00291                         }
00292                     }
00293                 }
00294
00295               /* CONDITION FOR CONVERGENCE */
00296               if(abs(last-edgepoints2) < 20)
00297                 nochange=0; /* means there is nochange stop */
00298               else
00299                 nochange =1; /* means there is change -> iterate */
00300               last = edgepoints2;
00301               count_points = count_points + edgepoints2;
00302               edgepoints2 =0;
00303             }
00304         }
00305
00306
00307
00308
00309
00310           printf(" Number of Edgepoints after hysterisis thresholding is %d\n",last );
00311 /*  if(count_points < 13000)
00312     {
00313       printf("============================================================\n");
00314       printf(" Reduce lower threshold or Upper threshold and try again \n");
00315       exit(0);
00316     }
00317 */
00318 printf(" Finsihed calculating the edges using thresholding and NMS\n");
00319
00320  /* WRITE THE IMAGE AFTER HYSTERISIS THRESHOLDING*/
00321
00322   if((fpy =fopen("edgemap_nms_hyst.pgm","w"))== 0)
00323         error(" Error writing file\n");
00324   fprintf(fpy, "P5\n%d %d\n255\n", ncols, nrows);
00325   for(i = 0; i < nrows; i++)
00326     if(fwrite(&edgemap[i][0], sizeof(char), ncols, fpy) != ncols)
00327       error("can't write the image");
00328
00329
00330   /* FOR EACH PIXEL IN MAGC IF IT HAS A LOCAL MAXIMA IN THE DIRECTION OF ANGLEC
00331      THEN IT IS AN EDGE */
00332
00333
00334
00335   for( i =0; i< nrows; i++)
00336     {
00337       for(j=0;j< ncols;j++)
00338         {
00339           angle = anglec[i][j];
00340
00341
```

```
00342           /* TO FIND POINTS FOR INTERPOLATION */
00343           /* printf("The angle is %f \n", angle); */
00344           /* BRUTE FORCE METHOD OF COMPARING EIGHT CASES */
00345
00346           if( edgemap[i][j] == 255 )
00347             {
00348
00349               anglecal = angle;
00350
00351               if(0 <= anglecal < M_PI/4)
00352                 {
00353                   r1=0;
00354                   c1=1;
00355                   r2 =1;
00356                   c2 =1;
00357                   r3 =0;
00358                   c3 =-1;
00359                   r4 = -1;
00360                   c4 -1;
00361                   /* mu = tan(angle); */
00362                 }
00363               if(M_PI/4 <= anglecal <M_PI/2)
00364                 {
00365                   r1=1;
00366                   c1=1;
00367                   r2 =1;
00368                   c2 =0;
00369                   r3 = -1;
00370                   c3 =-1;
00371                   r4 = -1;
00372                   c4 = 0;
00373                  /*  mu =1-1/tan(angle); */
00374                 }
00375               if(M_PI/2 <= anglecal < 3*M_PI/4)
00376                 {
00377                   r1=1;
00378                   c1=0;
00379                   r2 =1;
00380                   c2 =-1;
00381                   r3= -1;
00382                   c3 =0;
00383                   r4 = -1;
00384                   c4 =1;
00385                  /*  mu = -1/tan(angle); */
00386                 }
00387               if(3*M_PI/4 <= anglecal < M_PI)
00388                 {
00389                   r1=1;
00390                   c1=-1;
00391                   r2 =0;
00392                   c2 =-1;
00393                   r3 =-1;
00394                   c3 =1;
00395                   r4 = 0;
00396                   c4 =1;
00397                  /*  mu = -tan(angle); */
00398
00399                 }
00400
00401               if(M_PI <= anglecal < - M_PI/4)
00402                 {
00403                   r1=-1;
00404                   c1=1;
00405                   r2 =0;
00406                   c2 =1;
00407                   r3 =0;
00408                   c3 =-1;
00409                   r4 =1;
00410                   c4 = -1;
00411                   /* mu =tan(angle); */
00412                 }
00413               if(-M_PI/4 <= anglecal < -M_PI/2)
00414                 {
00415                   r1=-1;
00416                   c1=1;
00417                   r2 =-1;
00418                   c2 =0;
00419                   r3 = 1;
00420                   c3 =-1;
00421                   r4 =1;
00422                   c4 =0;
00423                  /*  mu =1-1/tan(angle); */
00424                 }
00425               if(-M_PI/2 <= anglecal < -3*M_PI/4)
00426                 {
00427                   r1=-1;
00428                   c1=0;
00429                   r2 =-1;
00430                   c2 =-1;
00431                   r3 = 1;
00432                   c3 =0;
00433                   r4=1;
00434                   c4 =1;
00435                   /* mu = -1/tan(angle); */
00436                 }
00437               if(-3*M_PI/4 <= anglecal < -M_PI)
00438                 {
00439                   r1=-1;
00440                   c1=-1;
00441                   r2 =0;
00442                   c2 =-1;
00443                   r3= 1;
00444                   c3 =1;
00445                   r4=0;
00446                   c4 =1;
00447                  /*  mu = -tan(angle); */
00448                 }
00449
00450               ivalue1 = LinearInterpolate(magc[i+r1][j+c1],magc[i+r2][j+c2],mu);
00451               ivalue2 =  LinearInterpolate(magc[i+r3][j+c3],magc[i+r4][j+c4],mu);
00452
00453               /* END OF COMPARING ANGLES */
00454
00455
00456               if( magc[i][j] > ivalue2 && magc[i][j] > ivalue2)
00457                 {
00458                   edgemap[i][j] = 255;
```

```
00459                   edgepoints1++;
00460
00461                 }
00462               else
00463                 {
00464                   edgemap[i][j] =0;
00465                 }
00466             }
00467         }
00468
00469
00470     }
00471
00472   /* PRINT IMAGE AFTER NMS */
00473       printf(" Number of Edgepoints after NMS on hysterisis thresholding is %d\n", edgepoints1 );
00474       /* WRITE THE IMAGE */
00475   if((fpy =fopen("edgemap_nms.pgm","w"))== 0)
00476     error(" Error writing file\n");
00477   fprintf(fpy, "P5\n%d %d\n255\n", ncols, nrows);
00478   for(i = 0; i < nrows; i++)
00479     if(fwrite(&edgemap[i][0], sizeof(char), ncols, fpy) != ncols)
00480       error("can't write the image");
00481   fclose(fpy);
00482
00483  /*  if((edgepoints1 > 12000 && edgepoints1 < 125000)) */
00484 /*     { */
00485 /*       printf(" # Edgepoints within the range specified\n"); */
00486 /*     } */
00487 /*   else if(edgepoints1 < 12000 ) */
00488 /*     { */
00489 /*       thresh = thresh -1; */
00490 /*       thresh2 = thresh2 +1; */
00491 /*       nms(anglec,magc,nrows,ncols,thresh,thresh2); */
00492 /*     } */
00493 /*   else */
00494 /*     { */
00495 /*       thresh = thresh + 1; */
00496 /*       thresh2 = thresh2 +1; */
00497 /*       nms(anglec,magc,nrows,ncols,thresh,thresh2); */
00498 /*     } */
00499 /*   printf("The lower threshold used = %d \n",thresh); */
00500 /*    printf("The Upper threshold used = %d \n",thresh2); */
00501 /*    printf("=========================================\n"); */
00502 /* } */
00503 }
00504
00505
00506 /*-------------------------------------------------------------------------*/
00507
00508 /* EDGE DETECTION BY ROBERTS OPERATOR */
00509
00510 /* The yc array is where the magnitude of the resultant correlation is stored*/
00511 /* The zc array is where the gradient of the resultant correlation is stored*/
00512 /* The edgec array is where edge/ not an edge info is stored in 1's or 0's */
00513
00514 void robert(unsigned char **xc, int nrows, int ncols, int thresh, int thresh2)
00515 {
00516
00517   float **row, **col,**theta,**y;
00518   unsigned char **edgemap;
00519   int i,j,edgepoints=0;
00520   FILE *fpy;
00521
00522   if((fpy =fopen("edgemap_roberts.pgm","w"))== 0)
00523     error(" Error writing file\n");
00524   printf("-------------------------------------------------------------------\n");
00525   printf(" Applying Robert Operator\n");
00526
00527   /* COMPUTE THE ROW COMPONENT */
00528    row = (float **)matrix(nrows, ncols, 0, 0, sizeof(float));
00529    col = (float **)matrix(nrows, ncols, 0, 0, sizeof(float));
00530    theta = (float **)matrix(nrows+2, ncols+2, -1, -1, sizeof(float));
00531    y = (float **)matrix(nrows+2, ncols+2, -1, -1, sizeof(float));
00532    edgemap = (unsigned char **) matrix(nrows,ncols,0,0,sizeof(char));
00533
00534    if( row == NULL || col == NULL || theta == NULL || y == NULL || edgemap == NULL)
00535      error(" Allocation error of matrices in Robet's sub-function\n");
00536
00537    printf("-----------------------------------------\n");
00538    printf(" Allocated temporary arrays\n");
00539    for(i = 0; i < nrows; i++)
00540      {
00541      for(j = 0; j < ncols; j++)
00542        {
00543          row[i][j]= (float)(xc[i][j] - xc[i-1][j-1])/sqrt(2);
00544          col[i][j]= (float)(xc[i-1][j]-xc[i][j-1])/sqrt(2);
00545          y[i][j] = row[i][j]*row[i][j]+ col[i][j]*col[i][j];
00546          theta[i][j]= atan2(col[i][j],row[i][j]);
00547        }
00548      }
00549
00550    /* CALL NMS BEFORE DOING ABSOLUTE THRESHOLDING */
00551    if(!skipNMS)
00552      nms(theta,y,nrows,ncols,thresh,thresh2);
00553
00554    /* DO THE THRESHOLDING TO COMPUTE THE EDGE PIXELS */
00555    for(i=0;i<nrows;i++)
00556      {
00557        for(j=0;j<ncols;j++)
00558          {
00559            if(y[i][j] >= thresh2*thresh2)
00560              {
00561                edgemap[i][j] = 255;
00562                edgepoints++;
00563              }
00564            else
00565              {
00566                edgemap[i][j] =0;
00567              }
00568          }
00569      }
00570
00571
00572    /* EDGE-POINTS */
00573    printf("-------------------------------------------------------------------\n");
00574    printf(" Number of Edgepoints using simple thresholding is %d\n",edgepoints);
00575    /* WRITE THE IMAGE */
```

```
00576   fprintf(fpy, "P5\n%d %d\n255\n", ncols, nrows);
00577   for(i = 0; i < nrows; i++)
00578     if(fwrite(&edgemap[i][0], sizeof(char), ncols, fpy) != ncols)
00579       error("can't write the image");
00580
00581
00582    /* CLOSE FILE & QUIT */
00583   fclose(fpy);
00584   exit(0);
00585
00586
00587
00588
00589 }
00590
00591 /*-------------------------------------------------------------------------*/
00592
00593 /* EDGE DETECTION BY PREWITS OPERATOR */
00594
00595 /* The yc array is where the magnitude of the resultant correlation is stored*/
00596 /* The zc array is where the gradient of the resultant correlation is stored*/
00597 /* The edgec array is where edge/ not an edge info is stored in 1's or 0's */
00598
00599 void prewit(unsigned char **xc, int nrows, int ncols, int thresh,int thresh2)
00600 {
00601   /* The yc array is where the magnitude of the resultant correlation is stored*/
00602 /* The zc array is where the gradient of the resultant correlation is stored*/
00603 /* The edgec array is where edge/ not an edge info is stored in 1's or 0's */
00604
00605
00606   float **row, **col,**theta,**y;
00607   unsigned char **edgemap;
00608   int i,j,edgepoints=0;
00609   FILE *fpy;
00610
00611   if((fpy =fopen("edgemap_prewitt.pgm","w"))== 0)
00612     error(" Error writing file\n");
00613   printf("-------------------------------------------------------------------\n");
00614   printf(" Applying Prewitt's Operator\n");
00615
00616   /* COMPUTE THE ROW COMPONENT */
00617    row = (float **)matrix(nrows, ncols, 0, 0, sizeof(float));
00618    col = (float **)matrix(nrows, ncols, 0, 0, sizeof(float));
00619    theta = (float **)matrix(nrows+2, ncols+2, -1, -1, sizeof(float));
00620    y = (float **)matrix(nrows+2, ncols+2, -1, -1, sizeof(float));
00621    edgemap = (unsigned char **) matrix(nrows,ncols,0,0,sizeof(char));
00622
00623    if( row == NULL || col == NULL || theta == NULL || y == NULL || edgemap == NULL)
00624      error(" Allocation error of matrices in Prewit's sub-function\n");
00625
00626    printf("-----------------------------------------\n");
00627    printf(" Allocated temporary arrays\n");
00628    for(i = 0; i < nrows; i++)
00629      {
00630      for(j = 0; j < ncols; j++)
00631        {
00632          row[i][j]= (float)(xc[i+1][j-1] +xc[i+1][j]+ xc[i+1][j+1] - xc[i-1][j-1]-xc[i-1][j]-xc[i-1][j+1])/6;
00633          col[i][j]= (float)(xc[i+1][j+1]+xc[i][j+1]+xc[i-1][j+1]-xc[i-1][j-1]-xc[i][j-1]-xc[i+1][j-1])/6;
00634          y[i][j] = row[i][j]*row[i][j]+ col[i][j]*col[i][j];
00635          theta[i][j]= atan2(col[i][j],row[i][j]);
00636        }
00637      }
00638
00639     /* CALL NMS BEFORE DOING ABSOLUTE THRESHOLDING */
00640
00641    if(!skipNMS)
00642      nms(theta,y,nrows,ncols,thresh,thresh2);
00643
00644
00645    /* DO THE THRESHOLDING TO COMPUTE THE EDGE PIXELS */
00646    for(i=0;i<nrows;i++)
00647      {
00648        for(j=0;j<ncols;j++)
00649          {
00650            if(y[i][j] >= thresh2*thresh2)
00651              {
00652                edgemap[i][j] = 255;
00653                edgepoints++;
00654              }
00655            else
00656              {
00657                edgemap[i][j] =0;
00658              }
00659          }
00660      }
00661
00662
00663    /* EDGE-POINTS */
00664    printf("-------------------------------------------------------------------\n");
00665    printf(" Number of Edgepoints using simple thresholding is %d\n",edgepoints);
00666
00667    /* WRITE THE IMAGE */
00668  fprintf(fpy, "P5\n%d %d\n255\n", ncols, nrows);
00669   for(i = 0; i < nrows; i++)
00670     if(fwrite(&edgemap[i][0], sizeof(char), ncols, fpy) != ncols)
00671       error("can't write the image");
00672
00673   /* CLOSE FILE & QUIT */
00674   fclose(fpy);
00675   exit(0);
00676
00677
00678
00679
00680 }
00681
00682 /*-------------------------------------------------------------------------*/
00683
00684 /* EDGE DETECTION BY SOBEL OPERATOR */
00685
00686 /* The yc array is where the magnitude of the resultant correlation is stored*/
00687 /* The zc array is where the gradient of the resultant correlation is stored*/
00688 /* The edgec array is where edge/ not an edge info is stored in 1's or 0's */
00689
00690 void sobel(unsigned char **xc, int nrows, int ncols, int thresh,int thresh2)
00691 {
00692   /* The yc array is where the magnitude of the resultant correlation is stored*/
```

```
00693 /* The zc array is where the gradient of the resultant correlation is stored*/
00694 /* The edgec array is where edge/ not an edge info is stored in 1's or 0's */
00695
00696
00697
00698   float **row, **col,**theta,**y;
00699   unsigned char **edgemap;
00700   int i,j,edgepoints=0;
00701   FILE *fpy;
00702
00703   if((fpy =fopen("edgemap_sobel.pgm","w"))== 0)
00704     error(" Error writing file\n");
00705   printf("-------------------------------------------------------------------\n");
00706   printf(" Applying Sobel Operator\n");
00707
00708   /* COMPUTE THE ROW COMPONENT */
00709    row = (float **)matrix(nrows, ncols, 0, 0, sizeof(float));
00710    col = (float **)matrix(nrows, ncols, 0, 0, sizeof(float));
00711    theta = (float **)matrix(nrows+2, ncols+2, -1, -1, sizeof(float));
00712    y = (float **)matrix(nrows+2, ncols+2, -1, -1, sizeof(float));
00713    edgemap = (unsigned char **) matrix(nrows,ncols,0,0,sizeof(char));
00714
00715    if( row == NULL || col == NULL || theta == NULL || y == NULL || edgemap == NULL)
00716      error(" Allocation error of matrices in Sobel's sub-function\n");
00717
00718    printf("-----------------------------------------\n");
00719    printf(" Allocated temporary arrays\n");
00720    for(i = 0; i < nrows; i++)
00721      {
00722      for(j = 0; j < ncols; j++)
00723        {
00724          row[i][j]= (float)(xc[i+1][j-1]+ 2*xc[i+1][j]+xc[i+1][j-1] - xc[i-1][j-1] - 2*xc[i-1][j] -xc[i-1][j+1])/8;
00725          col[i][j]= (float)(xc[i-1][j-1]+2 * xc[i][j-1]+xc[i+1][j-1] - xc[i-1][j+1] - 2*xc[i][j+1] - xc[i+1][j-1])/8;
00726          y[i][j] = row[i][j]*row[i][j]+ col[i][j]*col[i][j];
00727          theta[i][j]= atan2(col[i][j],row[i][j]);
00728        }
00729      }
00730
00731
00732     /* CALL NMS BEFORE DOING ABSOLUTE THRESHOLDING */
00733    if(!skipNMS)
00734      nms(theta,y,nrows,ncols,thresh,thresh2);
00735
00736
00737    /* DO THE THRESHOLDING TO COMPUTE THE EDGE PIXELS */
00738    for(i=0;i<nrows;i++)
00739      {
00740        for(j=0;j<ncols;j++)
00741          {
00742            if(y[i][j] >= thresh2*thresh2)
00743              {
00744                edgemap[i][j] = 255;
00745                edgepoints++;
00746              }
00747            else
00748              {
00749                edgemap[i][j] =0;
00750              }
00751          }
00752      }
00753
00754
00755    /* EDGE-POINTS */
00756    printf("-------------------------------------------------------------------\n");
00757    printf(" Number of Edgepoints using simple thresholding is %d\n",edgepoints);
00758
00759    /* WRITE THE IMAGE */
00760   fprintf(fpy, "P5\n%d %d\n255\n", ncols, nrows);
00761   for(i = 0; i < nrows; i++)
00762     if(fwrite(&edgemap[i][0], sizeof(char), ncols, fpy) != ncols)
00763       error("can't write the image");
00764
00765   /* CLOSE FILE & QUIT */
00766   fclose(fpy);
00767   exit(0);
00768
00769
00770
00771 }
00772
00773 /*-------------------------------------------------------------------------*/
00774
00775 /* EDGE DETECTION BY FRIE-CHEN OPERATOR */
00776
00777 /* The yc array is where the magnitude of the resultant correlation is stored*/
00778 /* The zc array is where the gradient of the resultant correlation is stored*/
00779 /* The edgec array is where edge/ not an edge info is stored in 1's or 0's */
00780
00781 void frie_chen(unsigned char **xc, int nrows, int ncols, int thresh,int thresh2)
00782 {
00783   /* The yc array is where the magnitude of the resultant correlation is stored*/
00784 /* The zc array is where the gradient of the resultant correlation is stored*/
00785 /* The edgec array is where edge/ not an edge info is stored in 1's or 0's */
00786
00787
00788   float **row, **col,**theta,**y;
00789   unsigned char **edgemap;
00790   int i,j,edgepoints =0;
00791   FILE *fpy;
00792
00793   if((fpy =fopen("edgemap_frie_chen.pgm","w"))== 0)
00794     error(" Error writing file\n");
00795   printf("-------------------------------------------------------------------\n");
00796   printf(" Applying Frie-Chen Operator\n");
00797
00798   /* COMPUTE THE ROW COMPONENT */
00799    row = (float **)matrix(nrows, ncols, 0, 0, sizeof(float));
00800    col = (float **)matrix(nrows, ncols, 0, 0, sizeof(float));
00801    theta = (float**)matrix(nrows+2, ncols+2, -1, -1, sizeof(float));
00802    y = (float **)matrix(nrows+2, ncols+2, -1, -1, sizeof(float));
00803    edgemap = (unsigned char **) matrix(nrows,ncols,0,0,sizeof(char));
00804
00805    if( row == NULL || col == NULL || theta == NULL || y == NULL || edgemap == NULL)
00806      error(" Allocation error of matrices in Frie-Chen's sub-function\n");
00807
00808    printf("-----------------------------------------\n");
00809    printf(" Allocated temporary arrays\n");
```

```
   for(i = 0; i < nrows; i++)
     {
     for(j = 0; j < ncols; j++)
       {
         row[i][j]=(float)(xc[i+1][j-1]+ sqrt(2)*xc[i+1][j]+xc[i+1][j-1] - xc[i-1][j-1] - sqrt(2)*xc[i-1][j] -xc[i-1][j+1])/8 ;
         col[i][j]= (float)(xc[i-1][j-1]+  sqrt(2)* xc[i][j-1]+xc[i+1][j-1] - xc[i-1][j+1] - sqrt(2)*xc[i][j+1] - xc[i+1][j-1])/8;;
         y[i][j] = row[i][j]*row[i][j]+ col[i][j]*col[i][j];
         theta[i][j]= atan2(col[i][j],row[i][j]);
       }
     }


    /* CALL NMS BEFORE DOING ABSOLUTE THRESHOLDING */
    if(!skipNMS)
      nms(theta,y,nrows,ncols,thresh,thresh2);


   /* DO THE THRESHOLDING TO COMPUTE THE EDGE PIXELS */
   for(i=0;i<nrows;i++)
     {
       for(j=0;j<ncols;j++)
         {
           if(y[i][j] >= thresh2*thresh2)
             {
               edgemap[i][j] = 255;
               edgepoints++;
             }
           else
             {
               edgemap[i][j] =0;
             }
         }
     }

/* EDGE-POINTS */
   printf("----------------------------------------------------------------\n");
   printf(" Number of Edgepoints using simple thresholding is %d\n",edgepoints);



   /* WRITE THE IMAGE */
 fprintf(fpy, "P5\n%d %d\n255\n", ncols, nrows);
  for(i = 0; i < nrows; i++)
    if(fwrite(&edgemap[i][0], sizeof(char), ncols, fpy) != ncols)
      error("can't write the image");

  /* CLOSE FILE & QUIT */
  fclose(fpy);
  exit(0);



}


/* ====================================================================================*/

int main(int argc, char **argv)
{
  FILE *fpx, *fpy;
  int nrows, ncols, i, j,thresh,thresh2;
  unsigned char **x;
  char *str;
  int r=0,p=0,s=0,f=0,histh=0;



  /* OPEN FILES */
  printf("-----------------------------------------------\n");
  printf("Opening image file\n");
  if(argc == 0) fpx = stdin;
  else
   {
     printf(" Making decesion\n");
     if(histh)
        {
          if((fpx = fopen(*(++argv), "r")) == NULL)
          {
                  printf("%s1\n",(*argv));
                  error("can't open file");
          }
        }
        else
        {
          if((fpx = fopen((str), "r")) == NULL)
          {
             printf("%s2\n",(*argv));
             error("can't open file");
          }
        }
   }
  printf("-----------------------------------------\n");
  printf(" Opened file --image file %s \n", *argv);
  fpy = stdout;

  /* READ HEADER */
  if(read_pgm_hdr(fpx, &nrows, &ncols) < 0)
    error("not a PGM image or bpp > 8");
  printf("------------------------------------------\n");
  printf(" Read Header \n");

  fclose(fpx);
  return 1;
}

/*/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Generated on Wed May 4 18:18:26 2005 for Edge detection by doxygen 1.3.6