CE 503 Photogrammetry I, Fall 2004
Homework 3(b)
Relative Orientation, Pairwise Rectification, and Anaglyph Presentation Using Small Format Camera
Assigned Friday 8 October
Due Wednesday 20 October

1. Find the file *503_04_hw3.zip* on the "geomatics" drive. That file will contain *ro1.m* (relative orientation) and *pr.c* (pairwise rectification), plus supporting functions and data. Take the data from measurement of your conjugate points from part (a) of this homework and enter them into the appropriate place in *ro1.m*. (See listing in following pages with critical sections marked in red).
2. Take the parameters and specifications for your camera and enter them into the appropriate places of *ro1.m*. Note that the file as delivered contains constants and values for the Nikon Coolpix 5700. Prior advice had been to scale the focal length in millimeters by 1/0.012 to convert to pixels. Based on survey of current cameras a better default value would be 1/0.003 to 1/0.006. CCD array sizes are often specified by the confusing "1/2 inch, 2/3 inch, etc.". These values are approximate dimensions for the *diagonal* of the array.
3. In MATLAB, run your modified program. If you are lucky, it will converge with small residuals and you can proceed to step 4. If it does not converge or if you have large residuals, then you have to try to get better camera constants or edit your observations. Consult instructor if you get stuck here. Your target should be convergence and max 3-5 pixel residuals.
4. Once you have convergence with modest sized residuals, then you can begin a "manual" approach to self-calibration for the three parameters: f, x0, y0. Start with f, then do x0, then y0. For each parameter, find the location where it is set in the file. Then by trial and error find the approximate value of the parameter which minimized the "misclosure" between the observations and the model. That misclosure is expressed in the number (sigma-nought hat, the *a posteriori* value of the reference standard deviation) that is printed to the screen with each iteration number. Use only the value from the last iteration. After you locate the approximate parameter value to minimize this misclosure, then bracket that value with a series of evenly spaced values, and produce and graph of sigma-nought vs. parameter value. It should look like a parabola in the vicinity of the minimum. See the samples presented in class on the notes section of the class site.
5. Once you have done this for f, then fix f at that value, and proceed to x0. When you are finished with x0, then fix it and proceed to y0. Fix y0 and make one final run of the program *r01.m*. The orientation values produced here will be the ones to use for the subsequent pairwise rectification.
6. Enter these values and other constants for your camera into the c-program file *pr.c*. Prepare your image files by changing mode from RGB color to grayscale, and save in *.raw* format (photoshop can do this, first in the "image" menu, then in the "file" menu). Enter the pathname for your files into *pr.c*. Make sure that *mat.h*

and *mat.c* are in the same folder as the main program *pr.c*. Then either import into visual studio, under VC++ as a "console application" and build, or use the supplied command line tools (*vcvars32.bat* and *nmake –f pr.mak*) to compile the source code into an executable. Run the produced *pr.exe* file to rectify and produce the anaglyph. Fiddle with the XMIN, YMAX, NOM_PLX variables to center the scene in your output image and to manage the absolute parallax. See instructor if this does not make sense.

7. Hand in (a) residual listing of your final relative orientation, (b) final parameters of interior and exterior orientation, (c) a description of your camera model, zoom setting, focal length, refined values, etc., (d) a hardcopy of your anaglyph scene, and an email copy. For cosmetic appeal you may wish to "black out" portions of the scene where the parallax is too great, there are non conjugate features, etc.

```
% ro1.m  -  6-oct-04
% relative orientation by collinearity

nrow=1920;
ncol=2560;
cl=[78;427;1121;1757;1909;33;1486;458;2082;2260;757;1043;72];
rl=[793;262;136;409;1696;1216;1528;1741;948;1507;1690;593;1687];
cr=[155;376;1092;1801;1901;97;1559;297;2297;2505;1089;990;113];
rr=[853;315;106;310;1715;1254;1531;1749;872;1494;1699;579;1698];

r0=nrow/2;
c0=ncol/2;

tcl=cl-c0;
trl=rl-r0;
tcr=cr-c0;
trr=rr-r0;

xl=tcl;
yl=-trl;
xr=tcr;
yr=-trr;

% units: feet, by pacing
Base=25;
Height=70;
B_H=Base/Height;

Xl=0.0;
Yl=0.0;
Zl=0.0;
Xr=Base;
Yr=0.0;
Zr=0.0;
alpha=atan((Base/2) / Height);
wl=0.0;
pl=-alpha;
kl=0.0;
wr=0.0;
pr=alpha;
kr=0.0;

phang_l=[wl;pl;kl];
phang_r=[wr;pr;kr];
phxyz_l=[Xl;Yl;Zl];
phxyz_r=[Xr;Yr;Zr];

% determined by brute force minimization
x0=7;
y0=-55;
foc=2590;

k1=0.0;
k2=0.0;
k3=0.0;
```

```
cam=[x0;y0;foc;k1;k2;k3];
load -ascii delta.dat

% make initial approximations for the obect points

[m,n]=size(cl);
npt=m;
X=zeros(npt,1);
Y=zeros(npt,1);
Z=zeros(npt,1);

for i=1:npt
  B=zeros(4,3);
  f=zeros(4,1);
  ndx=1;

  % first the left photo
  x=xl(i);
  y=yl(i);
  [c1,c2,f1,f2]=int_leq(x,y,phxyz_l,phang_l,cam);
  B(ndx:ndx+1,:)=[-1 0 c1; 0 -1 c2];
  f(ndx:ndx+1)=[f1;f2];
  ndx=ndx+2;

  % next the right photo
  x=xr(i);
  y=yr(i);
  [c1,c2,f1,f2]=int_leq(x,y,phxyz_r,phang_r,cam);
  B(ndx:ndx+1,:)=[-1 0 c1; 0 -1 c2];
  f(ndx:ndx+1)=[f1;f2];

  % ok now solve the LS intersection problem
  N=B'*B;
  t=B'*f;
  del=inv(N)*t;
  X(i)=del(1);
  Y(i)=del(2);
  Z(i)=del(3);
  end

% carry as unknowns: Yr,Zr,wr,pr,kr plus m-ground points
n=npt*2*2;
nu=5 + npt*3;
r=n-nu;

B=zeros(npt*2*2,nu);
f=zeros(m*2*2,1);
b=zeros(2,18);

converged=0;
threshold=1.0e-06;
prior_sighat=9.99e+09;
% ok now big iteration loop
for iter=1:10
  rndx=1;
  for i=1:npt
    ptxyz=[X(i);Y(i);Z(i)];
```

```
  % 2 equations from the left
  x=xl(i);
  y=yl(i);
  b=gencof(x,y,phang_l,phxyz_l,ptxyz,cam,delta);
  %disp('left');
  %keyboard

  cndx=5 + (i-1)*3 + 1;
  B(rndx,cndx)=b(1,15);
  B(rndx,cndx+1)=b(1,16);
  B(rndx,cndx+2)=b(1,17);
  f(rndx)=-b(1,18);
  B(rndx+1,cndx)=b(2,15);
  B(rndx+1,cndx+1)=b(2,16);
  B(rndx+1,cndx+2)=b(2,17);
  f(rndx+1)=-b(2,18);
  rndx=rndx+2;

  % now 2 equations from the right
  x=xr(i);
  y=yr(i);
  b=gencof(x,y,phang_r,phxyz_r,ptxyz,cam,delta);
  %disp('right');
  %keyboard
  B(rndx,1)=b(1,13);
  B(rndx,2)=b(1,14);
  B(rndx,3)=b(1,9);
  B(rndx,4)=b(1,10);
  B(rndx,5)=b(1,11);

  B(rndx,cndx)=b(1,15);
  B(rndx,cndx+1)=b(1,16);
  B(rndx,cndx+2)=b(1,17);
  f(rndx)=-b(1,18);

  B(rndx+1,1)=b(2,13);
  B(rndx+1,2)=b(2,14);
  B(rndx+1,3)=b(2,9);
  B(rndx+1,4)=b(2,10);
  B(rndx+1,5)=b(2,11);

  B(rndx+1,cndx)=b(2,15);
  B(rndx+1,cndx+1)=b(2,16);
  B(rndx+1,cndx+2)=b(2,17);
  f(rndx+1)=-b(2,18);
  rndx=rndx+2;
  end
N=B'*B;
t=B'*f;
del=inv(N)*t;
% now update the parameters
phxyz_r(2)=phxyz_r(2) + del(1);
phxyz_r(3)=phxyz_r(3) + del(2);
phang_r(1)=phang_r(1) + del(3);
phang_r(2)=phang_r(2) + del(4);
phang_r(3)=phang_r(3) + del(5);
```

```
  for i=1:npt
    ndx=5 + (i-1)*3 + 1;
    X(i)=X(i) + del(ndx);
    Y(i)=Y(i) + del(ndx+1);
    Z(i)=Z(i) + del(ndx+2);
    end

  v=f - B*del;
  sig2hat=(v'*v/r);
  sighat=sqrt(sig2hat);
  [iter sighat]

  d_sighat=abs(sighat - prior_sighat);
  if((d_sighat/sighat) < threshold)
    disp('we have converged');
    converged=1;
    break;
    end
  prior_sighat=sighat;
  % ok next iteration
  end

if(converged == 0)
  disp('NOT CONVERGED');
  end

disp('results');

disp('residuals');
v
disp('exp sta left / angles left');
phxyz_l'
phang_l'
disp('exp sta right / angles right');
phxyz_r'
phang_r'
disp('x0 y0 foc');
[x0 y0 foc]
```

```
/* pr.c  7-oct-04 */
/* pairwise rectify two images */
/* two nikon 5700 images of hovde hall */
/* for ce503 project demo */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "mat.c"

/* left 0299 */
#define OMEGA1   0.000000
#define PHI1    -0.176708856070037
#define KAPPA1   0.000
#define XL1      0.000
#define YL1      0.000
#define ZL1      0.000
#define F     2590.000

/* right 0300 */
#define OMEGA2   0.00934768146459455
#define PHI2     0.0501550232592291
#define KAPPA2  -0.0561980700711507
#define XL2     25.000
#define YL2     -1.76917871304201
#define ZL2     -3.29455515131471

#define INROW1 1920
#define INCOL1 2560
#define INROW2 1920
#define INCOL2 2560
#define OUTROW 1920
#define OUTCOL 2560

/*
#define XMIN -1280.000
#define YMAX   960.000
*/
#define XMIN  -550.000
#define YMAX  1150.000
#define STEP     1.000
#define NOM_PLX  675.0
#define ROW_SHIFT  960
#define COL_SHIFT 1280

unsigned char in_img1[INROW1][INCOL1];
unsigned char in_img2[INROW2][INCOL2];
unsigned char out_img[OUTROW][OUTCOL][3];

void main(void)
{
int i,j,k;
FILE *in_read,*out_write;
double **m1,**m2,**mb,**mn1,**mn2;
double **tmp1,**tmp2,**mn1t,**mn2t;
```

```
double **mx,**my,**mz, **m1t, **m2t;
double *uvw1,*uvw2,*vecl,*vecr;
double a01,a11,a21,b01,b11,b21;
double a02,a12,a22,b02,b12,b22;
double xp1,yp1,xp2,yp2;
double row1,col1,row2,col2;
int irow1,icol1,irow2,icol2,intest;
unsigned char red,green,blue;
double bx,by,bz,thx,thy,thz;
double p,w1,w2;
double xn,yn,xnl,xnr;
char ch;
double x0,y0;

m1=mat_alloc(3,3);
m2=mat_alloc(3,3);
m1t=mat_alloc(3,3);
m2t=mat_alloc(3,3);
mx=mat_alloc(3,3);
my=mat_alloc(3,3);
mz=mat_alloc(3,3);
mb=mat_alloc(3,3);
mn1=mat_alloc(3,3);
mn2=mat_alloc(3,3);
mn1t=mat_alloc(3,3);
mn2t=mat_alloc(3,3);
tmp1=mat_alloc(3,3);
tmp2=mat_alloc(3,3);

uvw1=vec_alloc(3);
uvw2=vec_alloc(3);
vecl=vec_alloc(3);
vecr=vec_alloc(3);

/* inner orientation 6-parameter transformation */

x0=7.0;
y0=55.0;

/* 0299 */

a01=y0;
a11=0.0;
a21=-1.0;
b01=x0;
b11=1.0;
b21=0.0;

/* 0300 */

a02=y0;
a12=0.0;
a22=-1.0;
b02=x0;
b12=1.0;
b22=0.0;
```

```
/* open and read the left image */
printf("open and read the left image\n");
if((in_read = fopen("i0299.raw","rb")) == NULL)
  {
  printf("cannot open left input image file\n");
  exit(1);
  }
/* read the left input image */
for(i=0; i<INROW1; i++)
  fread(in_img1[i],INCOL1,1,in_read);
fclose(in_read);

/* open and read the right image */
printf("open and read the right image\n");
if((in_read = fopen("i0300.raw","rb")) == NULL)
  {
  printf("cannot open right input image file\n");
  exit(1);
  }
/* read the left input image */
for(i=0; i<INROW2; i++)
  fread(in_img2[i],INCOL2,1,in_read);
fclose(in_read);

/* open the output file */
printf("open the output file\n");
if((out_write = fopen("pr.raw","wb")) == NULL)
  {
  printf("cannot open output image file\n");
  exit(1);
  }

ROTM(m1,OMEGA1,PHI1,KAPPA1);
ROTM(m2,OMEGA2,PHI2,KAPPA2);
for(i=1; i<=3; i++)
  {
  for(j=1; j<=3; j++)
    {
    m1t[i][j]=m1[j][i];
    m2t[i][j]=m2[j][i];
    }
  }

bx=XL2-XL1;
by=YL2-YL1;
bz=ZL2-ZL1;
thz=atan2(by,bx);
thy=atan2(-bz,sqrt(bx*bx+by*by));
/* extract tertiary omegas */
p= -asin(m1[1][3]);
w1=atan2(m1[2][3]/cos(p),m1[3][3]/cos(p));
p= -asin(m2[1][3]);
w2=atan2(m2[2][3]/cos(p),m2[3][3]/cos(p));
thx=(w1+w2)/2.0;

mx[1][1]=1.0;
mx[1][2]=0.0;
```

```
mx[1][3]=0.0;
mx[2][1]=0.0;
mx[2][2]=cos(thx);
mx[2][3]=sin(thx);
mx[3][1]=0.0;
mx[3][2]=-sin(thx);
mx[3][3]=cos(thx);

my[1][1]=cos(thy);
my[1][2]=0.0;
my[1][3]=-sin(thy);
my[2][1]=0.0;
my[2][2]=1.0;
my[2][3]=0.0;
my[3][1]=sin(thy);
my[3][2]=0.0;
my[3][3]=cos(thy);

mz[1][1]=cos(thz);
mz[1][2]=sin(thz);
mz[1][3]=0.0;
mz[2][1]=-sin(thz);
mz[2][2]=cos(thz);
mz[2][3]=0.0;
mz[3][1]=0.0;
mz[3][2]=0.0;
mz[3][3]=1.0;

MM(mx,my,3,3,3,tmp1);
MM(tmp1,mz,3,3,3,mb);
MM(mb,m1t,3,3,3,mn1);
MM(mb,m2t,3,3,3,mn2);
/* transpose */
for(i=1; i<=3; i++)
  {
  for(j=1; j<=3; j++)
    {
    mn1t[i][j]=mn1[j][i];
    mn2t[i][j]=mn2[j][i];
    }
  }

yn=YMAX;
for(i=0; i<OUTROW; i++)
  {
  printf("row %d\n",i);
  yn=yn - STEP;
  xn=XMIN;
  for(j=0; j<OUTCOL; j++)
    {
    xn=xn+STEP;
    xnl=xn;
    xnr=xnl - (NOM_PLX);
    vecl[1]=xnl;
    vecl[2]=yn;
    vecl[3]=-F;
    vecr[1]=xnr;
```

```
      vecr[2]=yn;
      vecr[3]=-F;
      Ab(mn1t,vecl,3,3,uvw1);
      Ab(mn2t,vecr,3,3,uvw2);
      xp1=-F*uvw1[1]/uvw1[3];
      yp1=-F*uvw1[2]/uvw1[3];
      xp2=-F*uvw2[1]/uvw2[3];
      yp2=-F*uvw2[2]/uvw2[3];
      row1=a01 + a11*xp1 + a21*yp1 + ROW_SHIFT;
      col1=b01 + b11*xp1 + b21*yp1 + COL_SHIFT;
      row2=a02 + a12*xp2 + a22*yp2 + ROW_SHIFT;
      col2=b02 + b12*xp2 + b22*yp2 + COL_SHIFT;
      /* make nearest neighbor interpolation */
      irow1=(int) (row1 + 0.5);
      icol1=(int) (col1 + 0.5);
      irow2=(int) (row2 + 0.5);
      icol2=(int) (col2 + 0.5);
      /* printf("%d %d\n",irow,icol); */
      /* ch=getchar(); */

      /* test limits left */
      intest=1;
      if((irow1 < 0) || (irow1 > (INROW1-1)))
        intest=0;
      if((icol1 < 0) || (icol1 > (INCOL1-1)))
        intest=0;
      if(intest == 1)
        red=in_img1[irow1][icol1];
      else
        red=150;

      /* test limits right */
      intest=1;
      if((irow2 < 0) || (irow2 > (INROW2-1)))
        intest=0;
      if((icol2 < 0) || (icol2 > (INCOL2-1)))
        intest=0;
      if(intest == 1)
        {
        green=in_img2[irow2][icol2];
        blue=green;
        }
      else
        {
        green=150;
        blue=green;
        }

      /* make the anaglyph stereo output pixel */

      out_img[i][j][0]=red;
      out_img[i][j][1]=green;
      out_img[i][j][2]=blue;
      }
    }
  for(i=0; i<OUTROW; i++)
    fwrite(out_img[i],OUTCOL,3,out_write);
```

```
fclose(out_write);
}
```