ATRテクニカルレポート表紙

[非公開]

TR-M-0039

動画像からのパノラマ映像生成に関する研究

山口　晃一郎　　朴　鍾一　　岩舘　祐一

Kouichirou YAMAGUCHI　　Jong-Il PARK　　Yuichi IWADATE

1998. 8.28

ATR 知能映像通信研究所

動画像からのパノラマ映像生成に関する研究
Creating Cylindrical Panorama from Video

ATR 知能映像通信研究所 第 3 研究室
学外実習生
山口 晃一郎

1998 年 9 月 1 日

1 はじめに

本研究の目的は、動画像よりパノラマ映像を生成することである。ビデオカメラから得られた画像群をつなぎ合わせることによりパノラマ画像を生成する。本研究では、ビデオカメラの位置を変えずに360°回転させ全周囲の画像群を得る。その画像群をマッチングすることによりカメラの回転パラメータを求め、画像を円筒にマッピングすることで円筒パノラマ画像を生成する。

また、生成した円筒パノラマ画像より任意方向の画像を生成する。

2 カメラパラメータの取得

ブロックマッチングを行い、最初の画像からの回転パラメータと焦点距離を推定する。詳細は文献[1]を参照のこと。

この推定により1番目の画像からの回転マトリックスと焦点距離が得られる。得られた1番目の画像からi番目の画像への変換をR_i、i番目の画像の焦点距離をF_iと表す。

3 円筒へのマッピング

ビデオカメラでの撮影より得られたn枚の画像を画像1から画像nとする。本手法ではこれらの画像群より円筒パノラマ画像を作成する。まず、すべての画像を同じ3次元座標系で表現し、それぞれの画像を半径F_1の円筒表面に投影する。投影された円筒表面を切り開いて平面にしたものが円筒パノラマ画像となっている。

したがって、円筒パノラマ画像を生成するためには図1に示されるようなi番目の画像の点からそれに対応する円筒パノラマ画像の点への変換を行う必要がある。以降、i番目の画像を画像iと表す。

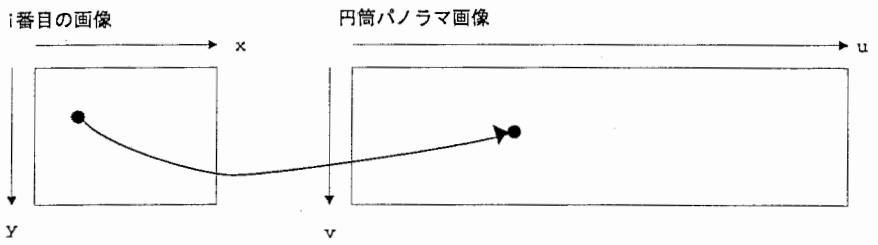

図1: (x,y)から(u,v)への変換

まず、画像iにおける3次元座標を、画像の中心が$(0,0,F_i)$となる図2のように定義する。F_iは画像iでのカメラの焦点距離。画像iの左上を$(0,0)$とする2次元座標(x,y)をこの3次元座標で表すと、

$$\left(x-\frac{S_x}{2}, y-\frac{S_y}{2}, F_i\right)$$

S_x, S_yはそれぞれ画像の横と縦の長さ。

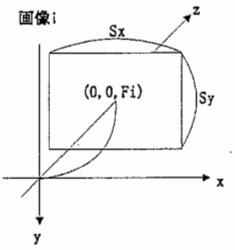

図 2: 画像 i の 3 次元座標

次に画像iの3次元座標で表された点を画像1での3次元座標に変換する。回転パラメータの推定で得られた画像1から画像iへの回転パラメータはR_iである。画像1の3次元座標から画像iの3次元座標への変換がR_iで表されることになるので、画像iから画像1への変換はR_i^{-1}で表される。よって、画像iの3次元座標で$\left(x-\frac{S_x}{2}, y-\frac{S_y}{2}, F_i\right)$の点を画像1の3次元座標で表した点$(X,Y,Z)$は

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = R_i^{-1} \begin{pmatrix} x-\frac{S_x}{2} \\ y-\frac{S_y}{2} \\ F_i \end{pmatrix}$$

となる。

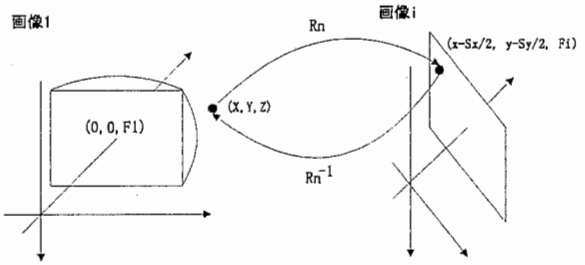

図 3: 画像 i から画像 1 の 3 次元座標への変換

画像1での3次元座標において、点$P(X,Y,Z)$を中心$(0,0,0)$半径F_1の円筒の表面に投影した点の円筒座標(θ,v)に変換する。図4の様に点Pを円筒表面に投影した点をQ、点Pからxz平面に下ろした垂線の足をHとすると、図5の様にθが表され、vは点Qのy座標だから図6より、

$$\begin{aligned} \theta &= \tan^{-1}\left(\frac{X}{Z}\right) \\ v &= \frac{F_1}{\sqrt{X^2+Z^2}} \end{aligned}$$

となる。

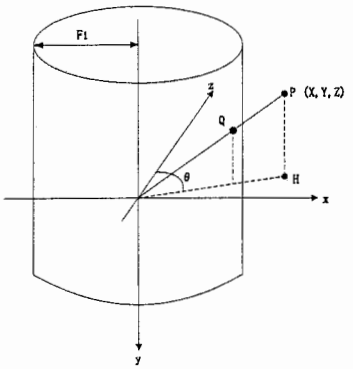

図 4: 円筒への投影

円筒表面を平面にした(u,v)平面では、uは図5での角度θの円周にあたるので、以下のようになる。

$$\begin{aligned} u &= 2\pi F_1 \times \frac{\theta}{2\pi} = F_1\theta \\ v &= \frac{F_1}{\sqrt{X^2+Z^2}} \end{aligned}$$

以上から画像iの点(x,y)を円筒パノラマ画像の点(u,v)に変換することができる。これにより、すべての画像のすべての点を円筒パノラマ画像へ変換することができ、円筒パノラマ画像を生成することが可能となる。

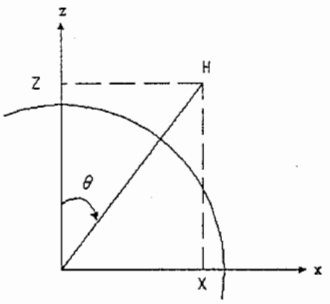

図 5: xz 平面図

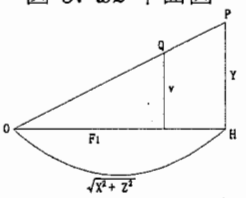

図 6: PHO で切った平面図

4 累積誤差の軽減

連続する画像間でブロックマッチングを行っているため、多くの画像から円筒画像を作成するとき、最初の画像と最後の画像の間ではそれまでの誤差が累積されずれが大きくなるという問題が起こる。本手法ではその累積誤差を小さくするため、最初の画像を最後にも付け加えて最初と最後の画像を同じものとし、それを一致させるようにすることで誤差を減らす。

4.1 焦点距離の変更

最初の画像の中心と最後の画像の中心を (u, v) 平面に変換したときの u のずれ ε を計算し、それにより新しい焦点距離を求める。まず最初の画像中心は $(u, v) = (0, 0)$ の点に変換される。円筒を開いた画像なので u の範囲は $2\pi F_1$ までで、円筒にしたとき $u = 0$ と $u = 2\pi F_1$ は重なることになる。図 7 の上の図のように画像 2,3$\cdots$ と進むにしたがってその画像の中心の u 座標が $2\pi F_1$ から小さくなっていき、最後の画像の中心が $u = 0$ まで行かなかった場合は、画像の横の長さ $2\pi F_1$ を小さくしてやることで、最初と最後の画像の中心を一致させようとする。逆に、図 7 の下のように画像 2,3$\cdots$ と進むにしたがってその画像の中心の u 座標が $2\pi F_1$ から小さくなっていき、最後の画像の中心が $u = 0$ を越えて再び $2\pi F_1$ より少し小さいところまで行った場合、画像の横の長さ $2\pi F_1$ を大きくする必要がある。2π は定数なので、F_1 をこれに従うように変化させる。

ずれ ε は図 7 に示されているように、最後の中心が最初の中心まで行かなかった場合は負の値、最後の中心が最初の中心を越えて行きすぎた場合は正の値を取るようにする。

ずれの角度 θ_e とすると、全体の横の長さは $2\pi F_1$ なので

$$\theta_e = 360^\circ \times \frac{\varepsilon}{2\pi F_1}$$

となる。

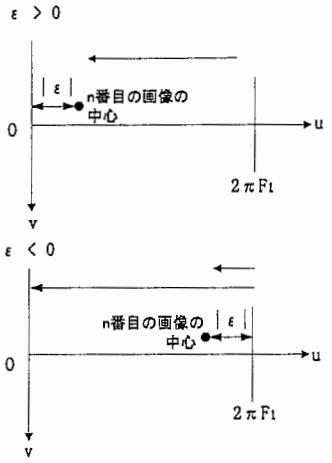

図 7: ずれの距離 ε

よって、新しい焦点距離 F_1' を

$$F_1' = \frac{360° - \theta_e}{360°} \times F_1$$

として、回転パラメータを求め直す。

これを、ずれ $|\varepsilon|$ が 1 以下になるまで繰り返す。

4.2 回転パラメータの変更

図 8 のように最初の画像から最後の画像への回転パラメータ R_n が累積された回転誤差を表している。R_n が単位行列 I になるとき最初の画像と最後の画像が一致するので、画像系列全体に R_n^{-1} を分散させてかける。つまり、

$$R_n^{-1} = (\delta r)^{(n-1)}$$

となる δr を求めて、回転パラメータ R_i を R_i' に変更する。

$$R_i' = (\delta r)^{(i-1)} R_i \qquad (2 \leq i \leq n)$$

これにより回転の誤差を全体に分散させて修正する。

しかし、δr を求めることは困難なので R_i^{-1} のそれぞれの軸の回転角度 α, β, γ を求め、この回転を全体に分割してかけている。

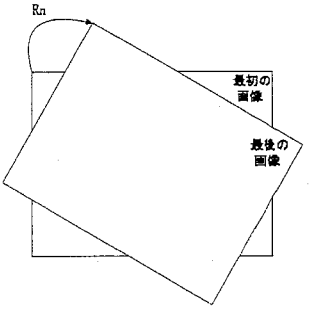

図 8: 回転のずれ

5 マッピング

5.1 v の範囲決定

図 9 の様に、すべての画像に対して画像の四隅の点と真ん中の上下 2 点を (u,v) 平面に変換して v の取り得る値の範囲を調べる。それにより、円筒画像の大きさを決定する。

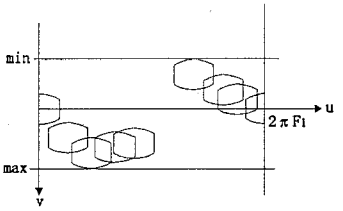

図 9: v の範囲

5.2 インデックスマップ作成

多くの画像から円筒画像を作成するとき、複数の画像で重なる部分が生じる。そのことからも、その点にもっとも近い中心を持つ画像によってマッピングを行うこととする。図 10 では、$i-1$、i、$i+1$ 番目の画像を円筒画像に変換したとき重なっている。中心がもっとも近い画像によりマッピングを行うので $i+1$ と i の中心の垂直 2 等分線と $i-1$ と i の中心の垂直 2 等分線に囲まれた色のついた部分が画像 i によってマッピングされる部分となる。

そのために、円筒画像の点がどの画像によりマッピングされるかというインデックスマップを作成する。図 11 のようにわけられることになる。図 11 で黒い四角の点が画像の中心を表し、その中心のまわりと同じ色の部分はその中心を持つ画像によりマッピングされることになる。

インデックスマップの作成は、まずすべての画像の中心を (u,v) 平面に変換する。そしてすべての (u,v) に対して、それに最も近い中心を持つ画像を決定する。

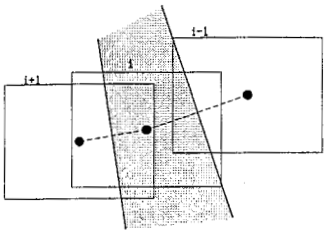

図 10: マッピングをする画像の決め方

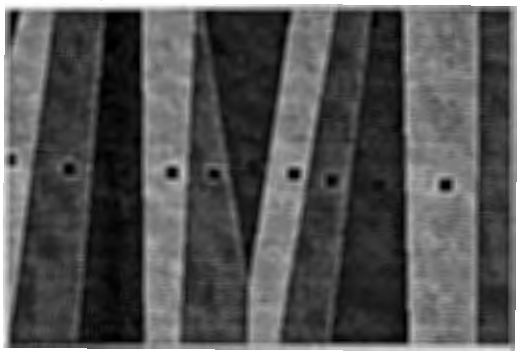

図 11: インデックスマップ

5.3 マッピング

それぞれの画像に対してその画像が (u,v) 平面で取り得る範囲を決定し、その範囲の (u,v) でインデックスマップの値がその画像の番号である点はその画像により画素値を決める。

円筒画像の画素の値を決定するにあたり、それぞれの画像の (x,y) から円筒画像 (u,v) へ変換を行った場合その (u,v) は整数値になるとは限らないので、その場合正数の (u,v) での画素の値を決め方が問題となる。そこで、画素値の決定は図 12 のように逆変換を行い整数の (u,v) から (x,y) に変換をする。(x,y) が整数とならないときにはその回りの 4 点から線形補間を行うことで画素値を決定する。

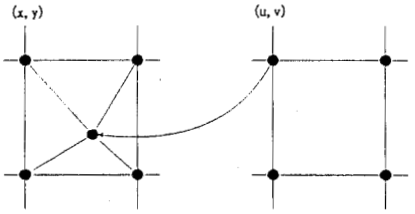

図 12: (u,v) から (x,y) への逆変換

6　任意方向画像生成

画像1の座標系で回転と焦点距離を指定して任意の投影面を図13の様に指定することができる。回転と焦点距離が分かればこれまでの円筒画像を生成したのとは逆の変換を行うことで円筒画像から任意方向の画像を生成することができる。

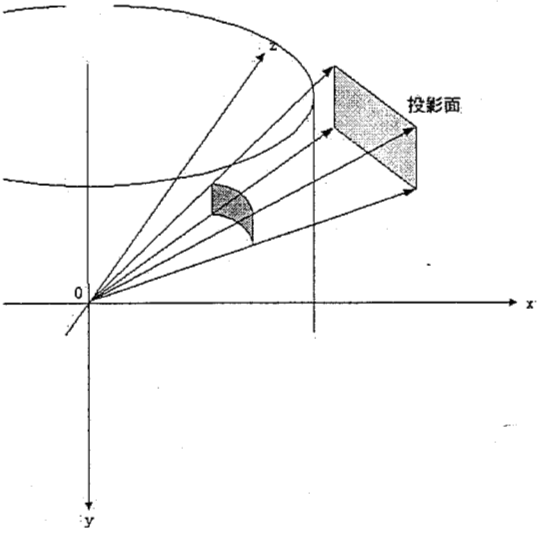

図 13: 投影面への投影

7　実験

実験では80枚の画像を用いた。図14が実験画像から生成した円筒画像である。大きなずれのない円筒パノラマ画像が生成された。

図 14: 円筒パノラマ画像

この円筒画像を用いて生成された任意方向の画像が図15で、その焦点距離を短くしたものが図16で、逆に焦点距離を長くしたものが図17である。また、図15から回転をさせて別の方向にしたものが図18である。

8　まとめ

今回の実験では100枚程度の画像について大きなずれなくパノラマ画像を生成できた。実験では単純に360°回転させた動画像を用いたので、垂直方向の範囲を増やすような回転を加えた場合についても実験を行い、それに対応させることも考えられる。

図 15: 任意方向画像

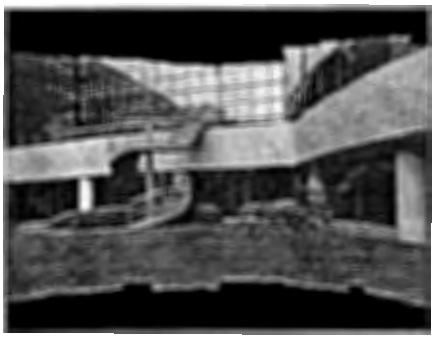

図 16: 焦点距離を短くした画像

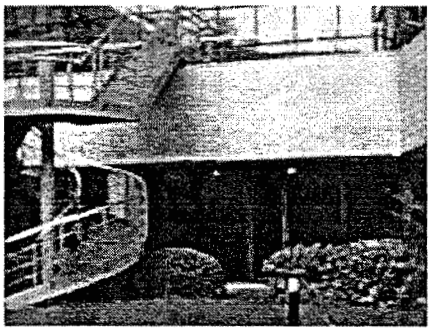

図 17: 焦点距離を長くした画像

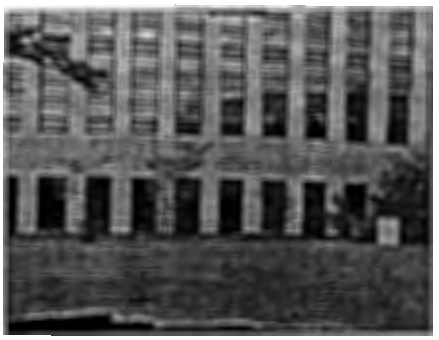

図 18: 回転させた画像

また、今回は動く物体を含む場合には対応していないが、回転パラメータを求める部分については動く物体を含む場合に対応しているので、その場合でもパノラマ画像を作成できるようにするという課題もある。

謝辞

本実習に置いて多大な御指導をいただいた朴鍾一研究員に心から感謝の意を表します。また、数々の助言やご支援をいただきました知能映像通信研究所第三研究室の方々に、深く感謝いたします。また、私に貴重な体験の場を提供して下さいましたATR知能映像通信研究所ならびに奈良先端科学技術大学院大学の関係諸氏にこの場を借りて厚く御礼申し上げます。

参考文献

[1] Jong-Il Park, Choong Woong Lee, "Robust estimation of camera parameters from image sequence for viedeo composition", Signal Processing: Image Communication 9 (1996)43-53

[2] Shmuel Peleg, Joshua Herman, "Panoramic Mosaics by Manifold Projection", CVPR'97

[3] Richard Szeliski, Heung-Yeung Shum, "Creating Full View Panoramic Image Mosaics and Environment Maps", SIGGRAPH'97 COMPUTER GRAPHICS Proceedings

A　プログラム

今回用いているのは以下のプログラムファイルである。

- mosaic2.c：円筒パノラマ画像生成メインプログラム
- mosfunc.c：円筒パノラマ画像生成の変換などを行う関数
- img_utln.c：画像ファイルの入出力を行う関数
- img_flt.c：画像のフィルター関数
- nr_utl.c：配列の生成などの操作を行う関数
- resample.c：2つの画像間の変換などを行う関数
- decomp.c：回転行列から回転パラメータを求める関数
- bma.c, dsvd.c, checkrange.c, fpm.c, putrank.c, cpe.c, gpf.c, rbst.c, gpl.c, del_para.c：回転パラメータ推定に用いる関数
- msview.c：任意方向画像生成のメインプログラム
- mvfunc.c：任意方向画像生成の変換などを行う関数
- panorama_view.tcl：任意方向画像生成プログラムのGUI

そのうち、mosaic2.c、mosfunc.c、msview.c、mvfunc.c、panorama_view.tclを付録として添付する。

B　ファイルの所在

プログラムファイルおよびこのレポートファイルは
miris69: /usr/people/yamaguti/prg/
にすべて置いてある。

```
/*-------------------------------------------------------------------------*
  Generation of cylinderical video mosaic from video

  Usage:   mosaic2 seq_name start_frame last_frame frame_skip fixtype [AOV]

  Outputs: cylinderical image (panorama.ppm)
 *-------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "mosaic.h"
#include "img_utln.h"
#include "nr_utl.h"
#include "fpm.h"
#include "cpe.h"
#include "resample.h"
#include "mosfunc.h"

#define PI 3.14159265358979323846264338327950288419716939937511

float Sigma=1.0; /* Sigma of smoothing filter for feature point matching */
int Flt_Tap=7;   /* Size of smoothing kernel for feature point matching */
int ME_Method=/*BLOCK_MATCHING*/BLOCK_PLUS_GRAD/*GRADIENT*/;
int Sx,Sy;
int Py;
double Aspr;  /* Aspect ratio of a pixel [dx] (reference: dy=1.0) */
double Ox,Oy,Oyo;
double InitAOV=30.0;
int Csx,Csy,Cox,Coy;
int Num_Frame;
int Bm=5; /* Boundary margin to be cleared */
int Mar=10; /* Panorama's vertical margin */
int SR_H,SR_V;

void show_command(char *s)
{
   printf("USAGE: %s seq_name start_frame last_frame frame_skip fixtype [AOV]\n",s);
   printf("                                    fixtype(0:no,1:zoom fix,2:AOV setting)\
n",s);
   exit(1);
}
/*-------------------------------------------------------------------------*/
main(int argc, char *argv[])
{
   int i,j,itr,frame,ref_frame,tref_frame,nf,start_frame,end_frame,frame_skip;
   int sx,sy,tsx,tsy,tix,tiy,usx,uix;
   int imgtype,colortype,nfp,fixtype,merge_method;
   char name[200],tmpname[200];
   unsigned char **prev,**curr,**itmp,**cimg_rsmpl,**gimg_rsmpl;
   unsigned char ***color_canvas,**color_canvas_out,**color_panorama,
                 **gray_canvas,**cimg,**gimg,**img1,**img2,**dbg_img1;
   double *Fo,*Fn,***Ro,***Rn,*Fp,***Rp,df,**dR,**ir,**tr,F_large;
   struct icoord *coxy;
   double x,y,u,v,theta,P,Q,R;
   int m,n,k,width,height,filetype,refmin,Csymin;
   unsigned char **canvas,**img;

   double **Rtmp,***Rot,**Inv,Rs;
   double da,db,dc,**dr,scx,lcx,thetae,**idr,ds,newF,gap;
   double **center,prevx,midx,tmpx;
   double dist,tmpdist,distmin,xp,yp;
   int red,green,blue,**index2,indA,indB;

   FILE *fp;
```



```
#ifdef DEBUG
   char testname[200];
   unsigned char **tmpimg,**dbg_img0;
#endif

/* Input Processing */

   if(argc < 6 || argc > 7) show_command(argv[0]);
   start_frame = atoi(argv[2]);
   end_frame   = atoi(argv[3]);
   frame_skip  = atoi(argv[4]);
   Num_Frame   = (end_frame-start_frame)/frame_skip + 1;
   fixtype     = atoi(argv[5]);
   if(argc == 7)
        InitAOV = atof(argv[6]);

/* Motion Estimation for Each Frame */

   sprintf(name,"%s%03d",argv[1],start_frame);
   prev=image_read_to_gray(name,&sx,&sy,&imgtype);
   smooth_image(prev,sx,sy,Flt_Tap,Sigma);

   Sx = sx;
   Sy = sy;
   Ox = (double)(Sx/2);
   Oy = (double)(Sy/2);
   Oyo= Oy;
   Aspr = 4.0*Sy/(3.0*Sx); /* for image aspect ratio of 3:4 */

   printf("---------------------------------------");
   printf("---------------------------------------\n");
   printf("***** Local Motion Estimation for Each Frame *****\n");
   printf("---------------------------------------");
   printf("---------------------------------------\n");

   for(frame=start_frame+frame_skip;frame<=end_frame;frame+=frame_skip)
        {
                SR_H =100;
                SR_V =30;


                sprintf(name,"%s%03d",argv[1],frame);
                sprintf(tmpname,"%s.fps",name);
                if((fp=fopen(tmpname,"rt")) != NULL)
                  {
                    image_close(prev,sx,sy,GRAY);
                    prev=image_read_to_gray(name,&sx,&sy,&imgtype);
                    smooth_image(prev,sx,sy,Flt_Tap,Sigma);
                    continue;
                  }
                curr=image_read_to_gray(name,&sx,&sy,&imgtype);
                smooth_image(curr,sx,sy,Flt_Tap,Sigma);

                nfp=feature_point_matching(prev,curr,sx,sy,name,ME_Method);

                itmp=prev;
                prev=curr;
                curr=itmp;
                image_close(curr,sx,sy,GRAY);

                printf("Frame %3d :  %4d feature points\n",frame,nfp);
        }
   image_close(prev,sx,sy,GRAY);

    Ox -= (double)Bm;
```

```
    Oy -= (double)Bm;
    sx -= 2*Bm;
    sy -= 2*Bm;


  do {
    /* Camera Parameter Estimation */

    printf("---------------------------------------");
    printf("---------------------------------------\n");
    printf("***** Camera Parameter Estimation for Each Frame *****\n");

    Fo=dvector(1,Num_Frame);
    Ro=dtensor(1,Num_Frame,1,3,1,3);
    Fn=dvector(1,Num_Frame);
    Rn=dtensor(1,Num_Frame,1,3,1,3);

    cpe(argv,Fo,Ro,fixtype);

  for(frame=1;frame<=Num_Frame;frame++)
    {
      printf("%3d | F=%8.3f | %9.6f %9.6f %9.6f \n",
             frame,Fo[frame],Ro[frame][1][1],Ro[frame][1][2],Ro[frame][1][3]);
      printf("                | %9.6f %9.6f %9.6f \n",
             Ro[frame][2][1],Ro[frame][2][2],Ro[frame][2][3]);
      printf("                | %9.6f %9.6f %9.6f \n",
             Ro[frame][3][1],Ro[frame][3][2],Ro[frame][3][3]);
    }


    /* Get Canvas Size */

    ref_frame = 1;
    canvas_size(sx,sy,Fo,Ro,ref_frame,Num_Frame,coxy);

    Csx = (int)(2*PI*Fo[ref_frame]*Oyo/Aspr);

    canvas = cmatrix(0,Csy*3,0,Csx);
    clear_cmatrix(canvas,0,Csy*3,0,Csx);

    /* New Focal Length */
    cyimap(sx/2.0, sy/2.0, Fo[1], Fo[Num_Frame], Ro[Num_Frame], &lcx, &y);
    if (lcx < Csx/2)
      gap = lcx;
    else
      gap = lcx - Csx;
    thetae = 360*gap/Csx;
    newF = (360-thetae)*Fo[1]/360;

    InitAOV = 2*atan(1/newF)*180/PI;
    printf("new AOV : %f\n",InitAOV);
  }while(gap > 1.0 || gap < -1.0);


  for(frame=1;frame<=Num_Frame;frame++){
    printf("%3d | F=%8.3f | %9.6f %9.6f %9.6f \n",
      frame,Fo[frame],Ro[frame][1][1],Ro[frame][1][2],Ro[frame][1][3]);
    printf("                | %9.6f %9.6f %9.6f \n",
      Ro[frame][2][1],Ro[frame][2][2],Ro[frame][2][3]);
    printf("                | %9.6f %9.6f %9.6f \n",
      Ro[frame][3][1],Ro[frame][3][2],Ro[frame][3][3]);
  }


/* decrease accuulated errors */
  Inv = dmatrix(1,3,1,3);
```



```
  for(i=1;i<=3;i++)
    for(j=1;j<=3;j++)
      Inv[i][j] = Ro[Num_Frame][j][i];
  decomp_rot(Inv,&da,&db,&dc);
  dr = dmatrix(1,3,1,3);
  da = da / (Num_Frame - 1);
  db = db / (Num_Frame - 1);
  dc = dc / (Num_Frame - 1);
  for (frame = 2;frame <= Num_Frame;frame++){
    get_rot(da*(frame-1),db*(frame-1),dc*(frame-1),dr);
    tmpR = dmatrix(1,3,1,3);
    for (frame = 2;frame <= Num_Frame;frame++){
    get_rot(da*(frame-1),db*(frame-1),dc*(frame-1),dr);
    for(i=1;i<=3;i++)
      for(j=1;j<=3;j++)
        tmpR[i][j]=dr[i][1]*Ro[frame][1][j]+
                   dr[i][2]*Ro[frame][2][j]+
                   dr[i][3]*Ro[frame][3][j];
    for(i=1;i<=3;i++)
      for(j=1;j<=3;j++)
        Ro[frame][i][j] = tmpR[i][j];
  }

  for(frame=1;frame<=Num_Frame;frame++)
    {
      printf("%3d | F=%8.3f | %9.6f %9.6f %9.6f \n",
             frame,Fo[frame],Ro[frame][1][1],Ro[frame][1][2],Ro[frame][1][3]);
      printf("                | %9.6f %9.6f %9.6f \n",
             Ro[frame][2][1],Ro[frame][2][2],Ro[frame][2][3]);
      printf("                | %9.6f %9.6f %9.6f \n",
             Ro[frame][3][1],Ro[frame][3][2],Ro[frame][3][3]);
    }



#ifdef DEBUG
printf("Get difference\n");

for(frame=1;frame<=Num_Frame-1;frame++){
  sprintf(name,"%s%03d",argv[1],start_frame+frame-1);
  img1=image_read_crop(name,Bm,&sx,&sy,&imgtype);
  dbg_img1=copy_to_gray(img1,sx,sy,imgtype);

  sprintf(name,"%s%03d",argv[1],start_frame+frame);
  img2=image_read_crop(name,Bm,&sx,&sy,&imgtype);
  gimg=copy_to_gray(img2,sx,sy,imgtype);

  gimg_rsmpl=cmatrix(0,sy-1,0,sx-1);
  tmpimg=cmatrix(0,sy-1,0,sx-1);
  image_clear(gimg_rsmpl,sx,sy);
  image_clear(tmpimg,sx,sy);

  Rtmp = dmatrix(1,3,1,3);
  Rtmp[1][1] = Ro[frame+1][1][1]*Ro[frame][1][1]+Ro[frame+1][1][2]*Ro[frame][1][2]+Ro
[frame+1][1][3]*Ro[frame][1][3];
  Rtmp[1][2] = Ro[frame+1][1][1]*Ro[frame][2][1]+Ro[frame+1][1][2]*Ro[frame][2][2]+Ro
[frame+1][1][3]*Ro[frame][2][3];
  Rtmp[1][3] = Ro[frame+1][1][1]*Ro[frame][3][1]+Ro[frame+1][1][2]*Ro[frame][3][2]+Ro
[frame+1][1][3]*Ro[frame][3][3];

  Rtmp[2][1] = Ro[frame+1][2][1]*Ro[frame][1][1]+Ro[frame+1][2][2]*Ro[frame][1][2]+Ro
[frame+1][2][3]*Ro[frame][1][3];
  Rtmp[2][2] = Ro[frame+1][2][1]*Ro[frame][2][1]+Ro[frame+1][2][2]*Ro[frame][2][2]+Ro
[frame+1][2][3]*Ro[frame][2][3];
  Rtmp[2][3] = Ro[frame+1][2][1]*Ro[frame][3][1]+Ro[frame+1][2][2]*Ro[frame][3][2]+Ro
{frame+1][2][3]*Ro[frame][3][3];
```

```
  Rtmp[3][1] = Ro[frame+1][3][1]*Ro[frame][1][1]+Ro[frame+1][3][2]*Ro[frame][1][2]+Ro
[frame+1][3][3]*Ro[frame][1][3];
  Rtmp[3][2] = Ro[frame+1][3][1]*Ro[frame][2][1]+Ro[frame+1][3][2]*Ro[frame][2][2]+Ro
[frame+1][3][3]*Ro[frame][2][3];
  Rtmp[3][3] = Ro[frame+1][3][1]*Ro[frame][3][1]+Ro[frame+1][3][2]*Ro[frame][3][2]+Ro
[frame+1][3][3]*Ro[frame][3][3];

  resample(gimg,sx,sy,GRAY,gimg_rsmpl,sx,sy,0,0,
           Fo[frame],Fo[frame+1],Rtmp);

  diff_with_bias(dbg_img1,gimg_rsmpl,sx,sy,1.0,0.0,0.0,tmpimg);
  sprintf(tmpname,"dbg_%s.dif",name);
  image_write(tmpimg,sx,sy,tmpname,PGM);
  free_cmatrix(gimg_rsmpl,0,sy-1,0,sx-1);
  free_cmatrix(tmpimg,0,sy-1,0,sx-1);
  free_cmatrix(img1,0,sy-1,0,sx-1);
  free_cmatrix(img2,0,sy-1,0,sx-1);
  free_cmatrix(dbg_img1,0,sy-1,0,sx-1);
  free_cmatrix(gimg,0,sy-1,0,sx-1);
  free_dmatrix(Rtmp,1,3,1,3);
}
#endif


/* Parameter Setting w.r.t. a specified reference frame */

    canvas_size(sx,sy,Fo,Ro,1,Num_Frame,coxy);
    Csymin = Csy;
    refmin = 1;

  ir=dmatrix(1,3,1,3);
  tr=dmatrix(1,3,1,3);
  Rot=dtensor(1,Num_Frame,1,3,1,3);
  for (k = 2;k <= Num_Frame;k++){
    ref_frame = k;
    for(i=1;i<=3;i++)
    for(j=1;j<=3;j++)
      ir[i][j]=Ro[ref_frame][j][i];
    for(frame=1;frame<=Num_Frame;frame++){
        update_rot(Ro[frame],ir,tr);
        for(i=1;i<=3;i++)
          for(j=1;j<=3;j++)
            Rot[frame][i][j]=tr[i][j];
      }

    canvas_size(sx,sy,Fo,Rot,ref_frame,Num_Frame,coxy);
      if (Csy < Csymin){
        Csymin = Csy;
        refmin = k;
    }
  }
  free_dtensor(Rot,1,Num_Frame,1,3,1,3);

  ref_frame = refmin;
printf("Ref Frame: %d\n",ref_frame);
  for(i=1;i<=3;i++)
  for(j=1;j<=3;j++)
    ir[i][j]=Ro[ref_frame][j][i];
 for(frame=1;frame<=Num_Frame;frame++){
       update_rot(Ro[frame],ir,tr);
       for(i=1;i<=3;i++)
         for(j=1;j<=3;j++)
           Ro[frame][i][j]=tr[i][j];
    }
  free_dmatrix(ir,1,3,1,3);
```



```
  free_dmatrix(tr,1,3,1,3);

#ifdef DEBUG
   for(frame=1;frame<=Num_Frame;frame++)
     {
       printf("%3d | F=%8.3f | %9.6f %9.6f %9.6f \n",
              frame,Fo[frame],Ro[frame][1][1],Ro[frame][1][2],Ro[frame][1][3]);
       printf("                  | %9.6f %9.6f %9.6f \n",
              Ro[frame][2][1],Ro[frame][2][2],Ro[frame][2][3]);
       printf("                  | %9.6f %9.6f %9.6f \n",
              Ro[frame][3][1],Ro[frame][3][2],Ro[frame][3][3]);
     }
#endif

  /* Get canvas size */
  canvas_size(sx,sy,Fo,Ro,ref_frame,Num_Frame,coxy);

  Csx = (int)(2*PI*Fo[ref_frame]*Oyo/Aspr);

   printf("Canvas Size= %4d x %4d , %4d\n",
          Csx,Csy,Coy);

  canvas = cmatrix(0,Csy*3,0,Csx);
  for (i = 0;i < Csy;i++)
    for (j = 0;j < Csx;j++){
      canvas[      i][j] = 0;
      canvas[  Csy+i][j] = 0;
      canvas[2*Csy+i][j] = 0;
    }

  /* Make Index Map */
  center = dmatrix(1,Num_Frame,1,2);
  for (frame = 1;frame <= Num_Frame;frame++){
    cyimap(sx/2.0,sy/2.0,Fo[ref_frame],Fo[frame],Ro[frame],&x,&y);
    center[frame][1] = x;
    center[frame][2] = y - Coy;
  }

  printf("Make index map %dx%d\n",Csx,Csy);
  index2 = imatrix(0,Csx,0,Csy);
  for (i = 0;i <= Csx;i++)
    for (j = 0;j <= Csy;j++){
      index2[i][j] = 1;
      distmin = -1.0;
      for (frame =1;frame <= Num_Frame;frame++)
        if ((abs((i-(int)center[frame][1])) < sx) &&
            (abs((j-(int)center[frame][2])) < sy)){
          dist = sqr(i-center[frame][1])+sqr(j-center[frame][2]);
          tmpdist = sqr(i-Csx-center[frame][1])+sqr(j-center[frame][2]);
          if (tmpdist < dist) dist = tmpdist;
          tmpdist = sqr(i-center[frame][1]+Csx)+sqr(j-center[frame][2]);
          if (tmpdist < dist) dist = tmpdist;
          if (dist < distmin || distmin < 0){
            distmin = dist;
            index2[i][j] = frame;
          }
        }
    }

/* Write index image */
#ifdef DEBUG
  for (i = 0;i <= Csx;i++)
    for (j = 0;j <= Csy;j++){
      if ((index2[i][j] % 3) == 0){
        canvas[      j][i]=0;
        canvas[  Csy+j][i]=0;
```

```
        canvas[2*Csy+j][i]=255;
      } else if ((index2[i][j] % 3) == 1) {
        canvas[      j][i]=255;
        canvas[  Csy+j][i]=0;
        canvas[2*Csy+j][i]=0;
      }else{
        canvas[      j][i]=0;
        canvas[  Csy+j][i]=255;
        canvas[2*Csy+j][i]=0;
      }
    }
  for(i = 1;i <=Num_Frame;i++){
    indA = (int)center[i][1];
    indB = (int)center[i][2];
    for(j=0;j < 10;j++)
      for(k=0;k < 10;k++){
        canvas[      indB+k][indA+j]=0;
        canvas[  Csy+indB+k][indA+j]=0;
        canvas[2*Csy+indB+k][indA+j]=0;
      }
  }
  image_write(canvas, Csx, Csy, "index.ppm",PPM);
  printf("write index image\n");
  clear_cmatrix(canvas,0,Csy*3,0,Csx);
#endif

/* Mapping */
  printf("Mapping %dx%d\n",Csx,Csy);
  for (frame = 1;frame <= Num_Frame;frame++){
    sprintf(name,"%s%03d",argv[1],start_frame+frame-1);
    img = image_read_crop(name, Bm,&width, &height, &filetype);

    prepare_paras(sx,sy,Fo[ref_frame],Fo[frame],Ro[frame],&tsx,&tsy,&tix,&tiy,&usx,&u
ix);
    printf("%3d : %3dx%3d, %4d %4d  ; %3dx%3d, %4d %4d\n",frame,tsx,tsy,tix,tiy,usx,t
sy,uix,tiy);

    for (i = 0;i < tsx;i++)
      for (j = 0;j < tsy;j++)
        if (index2[i+tix][j+tiy] == frame){
          x = (double)(i+tix);
          y = (double)(j+tiy+Coy);

          cyfmap(x,y,Fo[ref_frame],Fo[frame],Ro[frame],&xp,&yp);

          color_blint(img,width,height,(float)xp,(float)yp,&red,&green,&blue);

          if (red == 0 && green == 0 && blue == 0)
            continue;

          canvas[      j+tiy][i+tix]=red;
          canvas[  Csy+j+tiy][i+tix]=green;
          canvas[2*Csy+j+tiy][i+tix]=blue;
        }
    if (uix != 0)
      for (i = 0;i < usx;i++)
        for (j = 0;j < tsy;j++)
          if (index2[i+uix][j+tiy] == frame){
            x = (double)(i+uix);
            y = (double)(j+tiy+Coy);

            cyfmap(x,y,Fo[ref_frame],Fo[frame],Ro[frame],&xp,&yp);

            color_blint(img,width,height,(float)xp,(float)yp,&red,&green,&blue);

            if (red == 0 && green == 0 && blue == 0)
```



```
              continue;
            canvas[      j+tiy][i+uix]=red;
            canvas[  Csy+j+tiy][i+uix]=green;
            canvas[2*Csy+j+tiy][i+uix]=blue;
          }
    image_close(img,width,height,filetype);
  }

/* Output */
printf("Output\n");
  image_write(canvas, Csx, Csy, "panorama.ppm",filetype);
}
```

```
#include "math.h"
#include "nr_utl.h"
#include "img_utln.h"
#include "mosaic.h"

#define PI 3.14159265358979323846264338327950288419716939937511

extern int Sx,Sy,Py;
extern double Aspr;
extern double Ox,Oy,Oyo;
extern int Csx,Csy,Cox,Coy;
extern int Num_Frame;

void    cyfmap( /* forward mapping */
   double x,
   double y,
   double Fi,
   double Fo,
   double **r,
   double *xp,
   double *yp)
{
   double P,Q,R,val;
   double theta,z;

   x=x*Aspr/Oyo;
   y=y/Oyo;

   theta = x / Fi;
   x = Fi*sin(theta);
   z = Fi*cos(theta);

   P=(r[1][1]*x+r[1][2]*y+r[1][3]*z)*Fo;
   Q=(r[2][1]*x+r[2][2]*y+r[2][3]*z)*Fo;
   R=r[3][1]*x+r[3][2]*y+r[3][3]*z;

   *xp=(P*Oyo)/(R*Aspr)+Ox;
   *yp=(Q*Oyo/R)+Oy;
}
void    cyimap( /* inverse mapping */
   double x,
   double y,
   double Fi,
   double Fo,
   double **r,
   double *xp,
   double *yp)
{
   double P,Q,R,val,theta,u,v;


   x=(x-Ox)*Aspr/Oyo;
   y=(y-Oy)/Oyo;

   P=r[1][1]*x+r[2][1]*y+r[3][1]*Fo;
   Q=r[1][2]*x+r[2][2]*y+r[3][2]*Fo;
   R=r[1][3]*x+r[2][3]*y+r[3][3]*Fo;

   theta = atan(P/R);
   if (R < 0.0) theta = theta+PI;
   else if (theta < 0.0) theta = theta+2*PI;

   u = Fi*theta;
   v = Q*Fi/sqrt(P*P+R*R);

   *xp=(u*Oyo)/Aspr;
```



```
   *yp=(v*Oyo);
}

void canvas_size( /* get canvas size */
   int sx,
   int sy,
   double *F,
   double ***R,
   int ref_frame,
   int nf,
   struct icoord coxy[])
{
   int frame;
   double x,y,xp,yp;
   double ymax,ymin;

   ymax=0.0;
   ymin=0.0;

   for(frame=1;frame<=nf;frame++)
     {
       x=0.0;
       y=0.0;

       cyimap(x,y,F[ref_frame],F[frame],R[frame],&xp,&yp);

       if(yp > ymax)
         ymax=yp;
       if(yp < ymin)
         ymin=yp;
/*#ifdef DEBUG
printf("%3d | (%lf,%lf) -> (%lf,%lf)\n",frame,x,y,xp,yp);
#endif*/

       x=0.0;
       y=sy-1.0;

       cyimap(x,y,F[ref_frame],F[frame],R[frame],&xp,&yp);

       if(yp > ymax)
         ymax=yp;
       if(yp < ymin)
         ymin=yp;
/*#ifdef DEBUG
printf("%3d | (%lf,%lf) -> (%lf,%lf)\n",frame,x,y,xp,yp);
#endif*/

       x=sx-1.0;
       y=0.0;

       cyimap(x,y,F[ref_frame],F[frame],R[frame],&xp,&yp);

       if(yp > ymax)
         ymax=yp;
       if(yp < ymin)
         ymin=yp;
/*#ifdef DEBUG
printf("%3d | (%lf,%lf) -> (%lf,%lf)\n",frame,x,y,xp,yp);
#endif*/

       x=sx-1.0;
       y=sy-1.0;

       cyimap(x,y,F[ref_frame],F[frame],R[frame],&xp,&yp);

       if(yp > ymax)
```

```
          ymax=yp;
        if(yp < ymin)
          ymin=yp;
/*#ifdef DEBUG
printf("%3d | (%lf,%lf) -> (%lf,%lf)\n",frame,x,y,xp,yp);
#endif*/

        x=sx/2;
        y=0.0;

        cyimap(x,y,F[ref_frame],F[frame],R[frame],&xp,&yp);

        if(yp > ymax)
          ymax=yp;
        if(yp < ymin)
          ymin=yp;
/*#ifdef DEBUG
printf("%3d | (%lf,%lf) -> (%lf,%lf)\n",frame,x,y,xp,yp);
#endif*/

        x=sx/2;
        y=sy-1;

        cyimap(x,y,F[ref_frame],F[frame],R[frame],&xp,&yp);

        if(yp > ymax)
          ymax=yp;
        if(yp < ymin)
          ymin=yp;
/*#ifdef DEBUG
printf("%3d | (%lf,%lf) -> (%lf,%lf)\n",frame,x,y,xp,yp);
#endif*/
/*
        x=sx/2;
        y=sy/2;

        cyimap(x,y,F[ref_frame],F[frame],R[frame],&xp,&yp);

        coxy[frame].x=(int)xp;
        coxy[frame].y=(int)yp;
*/
      }

    Csy=(int)(ymax-ymin+2);
    Coy=(int)ymin-1;
}

void prepare_paras(
    int sx,
    int sy,
    double Fi,
    double Fo,
    double **R,
    int *tsx,
    int *tsy,
    int *tix, /* upper-left coordinate */
    int *tiy,
    int *usx,
    int *uix
    )
{
    int frame;
    double x,y,xp,yp;
    double xmax,ymax,xmin,ymin;
    double xmax2,xmin2;
```



```
    xmax2=xmin2=0.0;

    x=0.0;
    y=0.0;

    cyimap(x,y,Fi,Fo,R,&xp,&yp);
    yp = yp - Coy;

    xmax=xp;
    xmin=xp;
    ymax=yp;
    ymin=yp;

    x=sx-1.0;
    y=0.0;

    cyimap(x,y,Fi,Fo,R,&xp,&yp);
    yp = yp -Coy;

    if(xp > xmax)
      xmax=xp;
    if(xp < xmin){
      xmax2 = Csx;
      xmin2 = xmin;
      xmax = xp;
      xmin=0.0;
    }
    if(yp > ymax)
      ymax=yp;
    if(yp < ymin)
      ymin=yp;

    x=0.0;
    y=sy-1.0;

    cyimap(x,y,Fi,Fo,R,&xp,&yp);
    yp = yp -Coy;

    if (xmax2 == 0.0) {
      if (xp > xmax){
        xmin = 0.0;
        xmin2 = xp;
        xmax2 = Csx;
      }else if(xp > xmax)
        xmax=xp;
      else if(xp < xmin)
        xmin=xp;
    }else
      if (xp < xmin2)
        xmin2=xp;
    if(yp > ymax)
      ymax=yp;
    if(yp < ymin)
      ymin=yp;

    x=sx-1.0;
    y=sy-1.0;

    cyimap(x,y,Fi,Fo,R,&xp,&yp);
    yp = yp - Coy;

    if (xmax2 == 0.0) {
      if(xp > xmax)
        xmax=xp;
      if(xp < xmin){
        xmax2 = Csx;
```

```
        xmin2 = xmin;
        xmax = xp;
        xmin=0.0;
      }
    }else
      if (xp > xmax && xp < Csx /2)
        xmax=xp;
    if(yp > ymax)
      ymax=yp;
    if(yp < ymin)
      ymin=yp;

    x=sx/2;
    y=0.0;

    cyimap(x,y,Fi,Fo,R,&xp,&yp);
    yp = yp - Coy;

    if(yp > ymax)
      ymax=yp;
    if(yp < ymin)
      ymin=yp;

    x=sx/2;
    y=sy-1.0;

    cyimap(x,y,Fi,Fo,R,&xp,&yp);
    yp = yp - Coy;

    if(yp > ymax)
      ymax=yp;
    if(yp < ymin)
      ymin=yp;

    *tsx=(int)(xmax-xmin+1);
    *tsy=(int)(ymax-ymin+1);
    *tix=(int)(xmin);
    *tiy=(int)(ymin);
    *usx=(int)(xmax2-xmin2+1);
    *uix=(int)(xmin2);
}

void put_canvas(
    unsigned char **imgi,
    int sx,
    int sy,
    int ctype,
    unsigned char **imgr, /* resampled image */
    int tsx,
    int tsy,
    int tix,
    int tiy,
    double Fi,
    double Fo,
    double **R)
{
    int i,j,ir,ig,ib,m,n;
    double x,y,xp,yp;

    if(ctype == GRAY)
    {   for(i=0;i<tsy;i++)
        for(j=0;j<tsx;j++)
        {
          x=j+tix;
          y=i+tiy;
```



```
          cyfmap(x,y,Fi,Fo,R,&xp,&yp);

          imgr[i][j]=(unsigned char)blint(imgi,sx,sy,(float)xp,(float)yp);
        }
    }
    else if(ctype == COLOR){
        for(i=0;i<tsy;i++)
        for(j=0;j<tsx;j++)
        {
          x=j+tix;
          y=i+tiy+Coy;

          cyfmap(x,y,Fi,Fo,R,&xp,&yp);

          color_blint(imgi,sx,sy,(float)xp,(float)yp,&ir,&ig,&ib);

          m = j+tix;
          n = i+tiy;
/*printf(" (%4d, %4d) -> (%4f, %4f) : %d,%d,%d\n",m,n,xp,yp,ir,ig,ib);*/
          if (ir == 0 && ig == 0 && ib == 0)
            continue;
          imgr[       n][m]=ir;
          imgr[  Csy+n][m]=ig;
          imgr[2*Csy+n][m]=ib;
        }
    }
}
```

```
/*--------------------------------------------------------------------------*
  Generation of arbitrary view image from cylinderical panoramic image

  Usage:   mosview input.ppm angle1 angle2 focal_length

  Outputs: view.ppm
 *--------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "mosaic.h"
#include "img_utln.h"
#include "nr_utl.h"
#include "cpe.h"
#include "mosfunc.h"
#include "mvfunc.h"

#define PI 3.14159265358979323846264338327950288419716939937511

int Sx,Sy;
double Aspr;
double  Ox,Oy,Oyo;
double   InitAOV;
int Csx,Csy,Coy;

void main(int argc,char *argv[]){
  double theta,phi,Fi;
  int filetype,Csy2,Coy2;
  unsigned char **in_img,**out_img,**img;
  int sx,sy,i,j,red,green,blue,new_ox;
  double x,y,xp,yp,**Rot,f;

  /* Input processing */
  if (argc != 5){
     fprintf(stderr,"%s input angle1 angle2 Focal_length\n",argv[0]);
     exit(1);
  }

  theta = atof(argv[2]);
  phi   = atof(argv[3]);
  f     = atof(argv[4]);

  /* read input image */
  in_img = image_read(argv[1],&Csx,&Csy,&filetype);

  /* calculate focal_length of input image */
  Fi = Csx / (2*PI);

  /* prepare output image */
  Sx = 320;
  Sy = 240;
  Ox = (double)(Sx/2);
  Oy = (double)(Sy/2);
  Oyo= Oy;
  Aspr = 4.0*Sy/(3.0*Sx); /* for image aspect ratio of 3:4 */

  Rot = dmatrix(1,3,1,3);
  get_rot(-1*phi*PI/180.0,theta*PI/180.0,0.0,Rot);

  img = cmatrix(0,3*Sy,0,Sx);
  clear_cmatrix(img,0,3*Sy,0,Sx);

  Fi = Fi*Aspr/Oyo;
```



```
  /* make output image */
  for(i = 0;i < Sy;i++)
    for(j = 0;j < Sx;j++){
      cyimap((double)j,(double)i,Fi,Fi*f,Rot,&xp,&yp);
      yp = yp + Csy/2.0;

      cy_color_blint(in_img,Csx,Csy,(float)xp,(float)yp,&red,&green,&blue);

      if (red == 0 && green == 0 && blue == 0)
        continue;
      img[     i][j]=red;
      img[  Sy+i][j]=green;
      img[2*Sy+i][j]=blue;
    }
  image_write(img, Sx, Sy, "view.ppm",filetype);

  free_cmatrix(img,0,3*Sy,0,Sx);
  free_dmatrix(Rot,1,3,1,3);
  exit(0);
}
```

```
#include "math.h"
#include "nr_utl.h"
#include "img_utln.h"
#include "mosaic.h"

#define PI 3.14159265358979323846264338327950288419716939937511

void    cyrotfmap( /* forward mapping */
   double x,
   double y,
   double F,
   double a,
   double *xp,
   double *yp)
{
   double P,Q,R;
   double theta,z,rad_a,theta2;

   rad_a = a*PI/180;

   theta = x / F;
   x = F*sin(theta);
   z = F*cos(theta);

   P=x;
   Q=cos(rad_a)*y+sin(rad_a)*z;
   R=(-1)*sin(rad_a)*y+cos(rad_a)*z;

   theta2 = atan(P/R);
   if (R < 0.0) theta2 = theta2+PI;
   else if (theta2 < 0.0) theta2 = theta2+2*PI;

   *xp = F*theta2;
   *yp = Q*F/sqrt(P*P+R*R);
}


void    cyrotimap( /* inverse mapping */
   double x,
   double y,
   double F,
   double a,
   double *xp,
   double *yp)
{
   double P,Q,R;
   double theta,z,rad_a,theta2;

   rad_a = a*PI/180;

   theta = x / F;
   x = F*sin(theta);
   z = F*cos(theta);

   P=x;
   Q=cos(rad_a)*y-sin(rad_a)*z;
   R=sin(rad_a)*y+cos(rad_a)*z;

   theta2 = atan(P/R);
   if (R < 0.0) theta2 = theta2+PI;
   else if (theta2 < 0.0) theta2 = theta2+2*PI;

   *xp = F*theta2;
   *yp = Q*F/sqrt(P*P+R*R);
}
```



```
void cysize(
   int Csy,
   double F,
   double a,
   int *ysize,
   int *ymin)
{
  double min,max,x,y,xp,yp;

  x = 0.0;
  y = -1*Csy/2.0;
  cyrotimap(x,y,F,a,&xp,&yp);
  min = yp;
  max = yp;

  x = 0.0;
  y = Csy/2.0;
  cyrotimap(x,y,F,a,&xp,&yp);
  if (yp > max)
    max = yp;
  if (yp < min)
    min = yp;

  x = PI*F;
  y = -1*Csy/2.0;
  cyrotimap(x,y,F,a,&xp,&yp);
  if (yp > max)
    max = yp;
  if (yp < min)
    min = yp;

  x = PI*F;
  y = Csy/2.0;
  cyrotimap(x,y,F,a,&xp,&yp);
  if (yp > max)
    max = yp;
  if (yp < min)
    min = yp;

  *ysize = (int)(max-min+3);
  *ymin  = (int)min;
}

void cy_color_blint( /* get color from cylinderical image */
        unsigned char **img,
        int sx,
        int sy,
        float x,
        float y,
        int *r,
        int *g,
        int *b)
{
   float   val,tx,ty,t1,t2,t3,t4,a1,a2,a3,a4;
   int     ix,iy,ixp;

   if(    (x < 0) || (x >= sx) ||
          (y <= 0) || (y >= sy-1) )
   {
     *r=*g=*b=0;
   }
   else
   {
        ix = (int)x;
        iy = (int)y;
        tx = x - ix;
```

```
        ty = y - iy;
        t1 = (1.0-tx)*(1.0-ty);
        t2 = tx*(1.0-ty);
        t3 = (1.0-tx)*ty;
        t4 = tx*ty;
        if (ix == (sx-1)) ixp = 0;
        else ixp = ix+1;
        a1=img[iy][ix];
        a2=img[iy][ixp];
        a3=img[iy+1][ix];
        a4=img[iy+1][ixp];

        if( ((int)a1*a2*a3*a4) == 0)
          {
              *r=*g=*b=0;
              return;
          }
        *r = (int)( t1*a1
                   +t2*a2
                   +t3*a3
                   +t4*a4+0.5);

        iy += sy;

        *g = (int)( t1*(float)img[iy][ix]
                   +t2*(float)img[iy][ixp]
                   +t3*(float)img[iy+1][ix]
                   +t4*(float)img[iy+1][ixp]+0.5);

        iy += sy;

        *b = (int)( t1*(float)img[iy][ix]
                   +t2*(float)img[iy][ixp]
                   +t3*(float)img[iy+1][ix]
                   +t4*(float)img[iy+1][ixp]+0.5);
    }
}
```

```
#! /usr/bin/wish

if {$argc != 1} {
  puts [format "Usage: %s input_file" $argv0]
  exit 1
}

set angle1 0
set angle2 0
set fl     1

frame .f1
frame .f1.image
frame .f1.panel

image create photo image1

label .f1.image.lab -image image1
pack .f1.image.lab

set infile $argv

frame .f1.panel.file
label .f1.panel.file.lab -text "Input File :"
entry .f1.panel.file.ent -width 30
.f1.panel.file.ent delete 0 end
.f1.panel.file.ent insert 0 $infile
.f1.panel.file.ent xview end
bind .f1.panel.file.ent <Return> {set infile [.f1.panel.file.ent get];execview}
pack .f1.panel.file.lab -side left
pack .f1.panel.file.ent -side left -expand yes -fill x
pack .f1.panel.file -anchor w

scale .f1.panel.angle1 -label Angle1 -from 0 -to 360 -length 400 \
    -orient horizontal -variable angle1 -width 10
pack .f1.panel.angle1
scale .f1.panel.angle2 -label Angle2 -from -180 -to 180 -length 400 \
    -orient horizontal -variable angle2 -width 10
pack .f1.panel.angle2
scale .f1.panel.focal -label "Focal Length" -from 0 -to 5 -length 400 \
    -orient horizontal -variable fl -width 10 -resolution 0.01
pack .f1.panel.focal

button  .f1.panel.view -text View -command execview
pack .f1.panel.view

pack .f1.image .f1.panel

button .exit -text exit -command exit
pack .f1 .exit -pady 5 -ipady 2m


proc execview {} {
  global infile angle1 angle2 fl

  catch [exec /home/yamaguti/mosaic/mosview $infile $angle1 $angle2 $fl]
  image1 configure -file "view.ppm"
}
```