Advanced Robot Navigation

1 Problem Description

In this Laboratory, our task is to convert a geometric distorted image to a front view image.
First, warp a projective distorted image to get the affinely rectified image. Then, from the warped image, produce corresponding metrically rectified image.

1.1 Affine Rectification

As described in the text, to reconstruct the affinely rectified image from the perspec-tively distorted image, we first calculate the vanishing line. Secondly, find the matrix that transforms the vanishing line to the infinity line. And, apply the transform to the distorted image. Then, we get the affinely rectified image.
From 2 vanishing points, we can calculate the vanishing line on the distorted image. Suppose 4 points (a, b, c, and d) on the image in fig.(1) are the vertexes of the rectangular. Then, the line, l_{ab} that passes a and b and the the line, l_{cd} that passes c and c will meet at a vanishing point, namely v_1. Also the line, l_{ac}, and the line, l_{bd} will cross at another vanishing point, v_2.

$$l_{ab} = a \times b \tag{1}$$
$$l_{cd} = c \times d \tag{2}$$
$$l_{ac} = a \times c \tag{3}$$
$$l_{bd} = b \times d. \tag{4}$$

$$v_1 = l_{ab} \times l_{cd} \tag{5}$$
$$v_2 = l_{ac} \times l_{bd}. \tag{6}$$

The vanishing line, l_{vanish} can be calculated from the 2 vanishing points.

$$l_{vanish} = v_1 \times v_2. \tag{7}$$

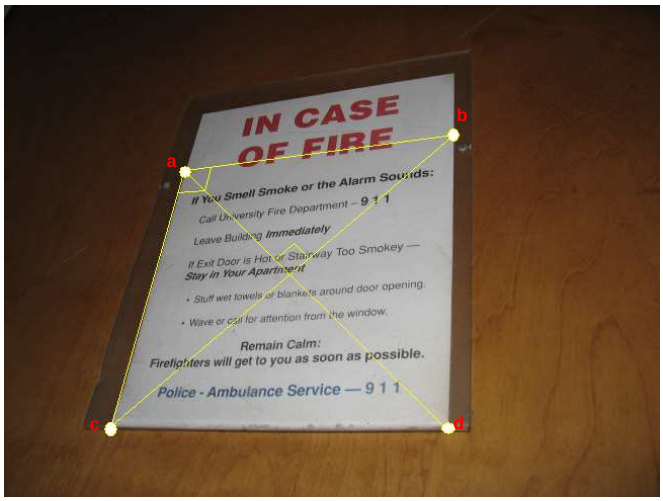

Figure 1: Geometrically distorted image.

Then. the affine rectification matrix, H' has following form.

$$H' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_{vanish}(1) & l_{vanish}(2) & l_{vanish}(3) \end{bmatrix}. \tag{8}$$

More generally, we can have the affine rectification matrix, H with any affine transform, H_a as described in the text.

$$H = H_a H'. \tag{9}$$

Applying H to the captured image, we finally can obtain the affinely rectified image.

1.2 Metric Rectification

2 pairs of the orthogonal lines gives us to calculate the affine transform that is applied in the affinely distorted image. Suppose the given 2 pairs of lines, (l_{ab}, l_{ac}) and (l_{ad}, l_{bc}) in fig.(1) are orthogonal respectively in the distortion free image. Then from the corresponding 2 pairs of lines (m_1, l_1) and (m_2, l_2) in affinely rectified image, we can get the affine transform as described in the class.

$$\begin{bmatrix} l(1) & l(2) \end{bmatrix} \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \begin{bmatrix} m(1) \\ m(2) \end{bmatrix} = 0. \tag{10}$$

Since the matrix, S, is symmetric and scale free, we only need 2 pairs of lines, (m_1, l_1) and (m_2, l_2) to calculate S.

Let $s_{22} = 1$ then we have following equation.

$$\begin{bmatrix} l_1(1)m_1(1) & l_1(2)m_1(1) + l_1(1)m_1(2) \\ l_2(1)m_2(1) & l_2(2)m_2(1) + l_2(1)m_2(2) \end{bmatrix} \begin{bmatrix} s_{11} \\ s_{12} \end{bmatrix} = \begin{bmatrix} -l_1(2)m_1(2) \\ -l_2(2)m_2(2)) \end{bmatrix}. \tag{11}$$

By solving this equation, we get the matrix S.

$$S = \begin{bmatrix} s_{11} & s_{12} \\ s_{12} & 1 \end{bmatrix}. \tag{12}$$

And $S = AA^T$ where A is the affine kernel that we want to get. From the lecture note, we can calculate A as follows

$$A = U \begin{bmatrix} \sqrt{\sigma_1} & 0 \\ 0 & \sqrt{\sigma_2} \end{bmatrix} V^T \tag{13}$$

,where

$$S = U \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} V^T. \tag{14}$$

Finally, we obtain the affine transform,

$$H = \begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}. \tag{15}$$

, where t is some shift parameter. Applying inverse of H, we can reconstruct the distortion free image (undo the affine transform).

2 Results

The captured image in fig.(2) shows the geometric(perspective)distortion. And we get the affinely rectified image in fig.(3) from the captured image by applying the method that is described above. Then, we finally obtain the metrically rectified image in fig.(4) from the affinely distorted image.

Figure 2: Perspectively distorted image.

Figure 3: Affine rectification result image.

3 Codes

The codes are composed of 3 files which are Lab2.cpp, rectification.cpp, and rectification.h. Lab2.cpp is the main function file. The program functions are described in rectification.cpp. And *opencv* library is used to implement most of the functions.

3.1 Main Function File : Lab2.cpp

```
//
// file : Lab2.cpp
//
------------------------------------------------------------------------//
This main program converts a geometric distorted image to a front view image.
```

Figure 4: Metric rectification result image.

```
// First, it warps a projective distorted image to get the affinely
// rectified image. Then, from the warped image, produces corresponding
// metrically rectified image.
//

#include "rectification.h"

int main() {
    // declaration
    Rectification   Rec;
    IplImage        *inImage = 0;
    IplImage        *outImage = 0;
    int             outHeight, outWidth, outChannels;
    char            inImageName[80];

    // load an image
    printf("Type input image name : ");
    scanf("%s", inImageName);
    inImage = cvLoadImage(inImageName, 1);
    if(!inImage){
        printf("Could not load image file: %s\n", inImageName);
        exit(0);
    }

    // create the output image
    outHeight   = inImage->height;
    outWidth    = inImage->width;
    outChannels = inImage->nChannels;
```

```
outImage    = cvCreateImage(cvSize(outWidth, outHeight),
              IPL_DEPTH_8U, outChannels);

//////////////////////////////////////////
//         affine rectification          //
//////////////////////////////////////////
// get the distorted image's 4 point positions

int inX1, inY1, inX2, inY2, inX3, inY3, inX4, inY4;
printf("\n Type the 1st point in the distorted image (x1, y1) : ");
scanf("%d, %d", &inX1, &inY1);
printf("\n Type the 2nd point in the distorted image (x2, y2) : ");
scanf("%d, %d", &inX2, &inY2);
printf("\n Type the 3rd point in the distorted image (x3, y3) : ");
scanf("%d, %d", &inX3, &inY3);
printf("\n Type the 4th point in the distorted image (x4, y4) : ");
scanf("%d, %d", &inX4, &inY4);

// homogeneous representation of the 4 given points in the distorted image
float aArr[3] = {inX1, inY1, 1};
float bArr[3] = {inX2, inY2, 1};
float cArr[3] = {inX3, inY3, 1};
float dArr[3] = {inX4, inY4, 1};

CvMat *a, *b, *c, *d;
a = cvCreateMat(3, 1, CV_32FC1);
b = cvCreateMat(3, 1, CV_32FC1);
c = cvCreateMat(3, 1, CV_32FC1);
d = cvCreateMat(3, 1, CV_32FC1);

Rec.Array2CvMat(aArr, a, 3, 1);
Rec.Array2CvMat(bArr, b, 3, 1);
Rec.Array2CvMat(cArr, c, 3, 1);
Rec.Array2CvMat(dArr, d, 3, 1);

// for affine rectification, we apply a matrix
// s.t. H = Ha*[1 0 0; 0 1 0; l1 l2 l3],
// where Ha is an affine transform and [l1 l2 l3] is the vanishing line.
// also, we can select any Ha for the affine rectification.
// so, following affine transform is chosen to fit the result image
// on the given window.

// affine transfom for affine rectification
float affineXform[9] = {2, .7, 0,
                        0,  2, 0,
```

```
                            0,  0, 1};
    CvMat Ha = cvMat(3, 3, CV_32FC1, affineXform);
    CvMat *H; // affine rectification matrix
    H = cvCreateMat(3, 3, CV_32FC1);
    Rec.AffineRectification(inImage, outImage, a, b, c, d, &Ha, H);

    ////////////////////////////////////////////
    //         metric rectification           //
    ////////////////////////////////////////////
    cvCopyImage (outImage, inImage);
    // pixel shift for the affine transform in metric rectification
    float shiftX = 70,  shiftY = 60;

    // calculate the corresponding 4 points
    CvMat *aa, *bb, *cc, *dd;
    // corresponding 4 points in affinely rectified image
    aa = cvCreateMat(3, 1, CV_32FC1);
    bb = cvCreateMat(3, 1, CV_32FC1);
    cc = cvCreateMat(3, 1, CV_32FC1);
    dd = cvCreateMat(3, 1, CV_32FC1);
    cvMatMul(H, a, aa); // x' = Hx, where H is the affine rectification matrix
    cvMatMul(H, b, bb);
    cvMatMul(H, c, cc);
    cvMatMul(H, d, dd);
    Rec.MetricRectification(inImage, outImage, aa, bb, cc, dd, shiftX, shiftY);

    // release matrices
    cvReleaseMat(&a); cvReleaseMat(&b); cvReleaseMat(&c); cvReleaseMat(&d);
    cvReleaseMat(&aa); cvReleaseMat(&bb); cvReleaseMat(&cc); cvReleaseMat(&dd);
    cvReleaseMat(&H);
    // release the images
    cvReleaseImage(&inImage);
    cvReleaseImage(&outImage);

    return 0;
}
```

3.2 Program Header File : rectification.h

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cv.h"
#include "cxcore.h"
```

```
#include "highgui.h"

#define IMPLEMENTATION  2
#define min(a, b)   ((a <= b) ? a : b)
#define max(a, b)   ((a >= b) ? a : b)

class Rectification {
public:
    void AffineRectification(IplImage *inImage, IplImage *outImage,
                             CvMat *a, CvMat *b, CvMat *c, CvMat *d,
                             CvMat *Ha, CvMat *H);
    void MetricRectification(IplImage *inImage, IplImage *outImage,
                             CvMat *a, CvMat *b, CvMat *c, CvMat *d,
                             float shiftX, float shiftY);
    void Array2CvMat(float *arr, CvMat *cvArr, int row, int column);
    void CvMat2Array(CvMat *cvArr, float *arr, int row, int column);

private:
    void CalculateVanishingLine(CvMat *a, CvMat *b, CvMat *c, CvMat *d,
                                CvMat *vanishLine);
};
```

3.3 Program Function File : rectification.cpp

```
#include "rectification.h"


void Rectification::AffineRectification(IplImage*inImage,IplImage*outImage,
                                        CvMat *a, CvMat *b, CvMat *c, CvMat *d,
                                        CvMat *Ha, CvMat *H)
{
    // get the distorted image's 4 point positions
    // homogeneous representation of the 4 given points in the distorted image

    uchar       *inData;
    uchar       *outData;
    int         inHeight, inWidth, inStep, inChannels;
    int         outHeight, outWidth, outStep, outChannels;
    int         i, j, k;
    char        outImageName[80];
    CvMat       *vanishLine;
    CvMat       *Hv2i, *invH; // matrices for affine rectification

    vanishLine  = cvCreateMat(3, 1, CV_32FC1);
```

```
Hv2i        = cvCreateMat(3, 3, CV_32FC1);
invH        = cvCreateMat(3, 3, CV_32FC1);

// get the input image data
inHeight    = inImage->height;
inWidth     = inImage->width;
inStep      = inImage->widthStep;
inChannels  = inImage->nChannels;
inData      = (uchar *)inImage->imageData;

// create the affinely rectified image
outHeight   = inHeight;
outWidth    = inWidth;
outChannels = inChannels;
outStep     = outImage->widthStep;
outData     = (uchar *)outImage->imageData;

// initialize result image
for(i = 0; i < outHeight; i++){
    for(j = 0; j < outWidth; j++){
        for(k = 0; k < outChannels; k++){
            outData[i*outStep + j*outChannels + k] = 0;
        }
    }
}

// get vanishing line
CalculateVanishingLine(a, b, c, d, vanishLine);

// vanish2Inf : vanish line to infinity line
float vl1 = cvmGet(vanishLine, 0, 0);
float vl2 = cvmGet(vanishLine, 1, 0);
float vl3 = cvmGet(vanishLine, 2, 0);
float vanish2Inf[9] = {1, 0, 0,
                       0, 1, 0,
                       vl1, vl2, vl3};

// Hv2i : vanish line to infinity line
Array2CvMat(vanish2Inf, Hv2i, 3, 3);
// H  = Ha * Hv2i : affine rectification matrix
cvMatMul(Ha, Hv2i, H);
cvInvert(H, invH);

// apply the transform to get the target image
// ------------------------------------------------
```

```
// out(x') = in(x) : has 2 implementation forms
// case 1 : out(Hx) = in(x)
// case 2 : out(x') = inv(H)x'
float h[9];
if(IMPLEMENTATION == 1){ // case 1 : out(Hx) = in(x)
    CvMat2Array(H, h, 3, 3);
}else{ // case 2 : x = inv(H)x'
    CvMat2Array(invH, h, 3, 3);
}

int ii, jj;
float x1, x2, x3;
for(i = 0; i < inHeight-3; i++){ // case 1: i, j: x, ii, jj: x', x' = Hx
    for(j = 0; j < inWidth; j++){// case 2: i, j: x', ii, jj: x, x = invHx'
        for(k = 0; k < inChannels; k++){ // x: distorted, x': undistorted
            x1 = h[0] * j + h[1] * i + h[2];
            x2 = h[3] * j + h[4] * i + h[5];
            x3 = h[6] * j + h[7] * i + 1;
            ii = min(outHeight - 1, max(0, (int)(x2 / x3)));
            jj = min(outWidth - 1, max(0, (int)(x1 / x3)));
            if(IMPLEMENTATION == 1) // case 1 : out(Hx) = in(x)
                outData[ii*outStep + jj*outChannels + k]
                = inData[i*inStep + j*inChannels + k];
            else // case 2 : out(x') = in(inv(H)x')
                outData[i*outStep + j*outChannels + k]
                = inData[ii*inStep + jj*inChannels + k];
        }
    }
}

// create windows
cvNamedWindow("input image", CV_WINDOW_AUTOSIZE);
cvNamedWindow("output image", CV_WINDOW_AUTOSIZE);

// show the images
cvShowImage("input image", inImage);
cvShowImage("output image", outImage);

// wait for a key
cvWaitKey(0);
cvDestroyWindow("input image");
cvDestroyWindow("output image");

// release matrices
cvReleaseMat(&vanishLine);
```

```
    cvReleaseMat(&Hv2i);
    cvReleaseMat(&invH);

    // write output image
    sprintf(outImageName, "result1.jpg");
    if(!cvSaveImage(outImageName, outImage)){
        printf("Could not save: %s\n",outImageName);
    }
}


void Rectification::MetricRectification(IplImage *inImage, IplImage *outImage,
                                        CvMat *a, CvMat *b, CvMat *c, CvMat *d,
                                        float shiftX, float shiftY)
{
    uchar       *inData;
    uchar       *outData;
    int         inHeight, inWidth, inStep, inChannels;
    int         outHeight, outWidth, outStep, outChannels;
    int         i, j, k;
    char        outImageName[80];
    CvMat       *m1, *l1, *m2, *l2; // 2 pairs of right angle lines
    CvMat       Mat, Vec, *S_Vec; // matrix and vectors to calculate S_Mat
    CvMat       *S, *U, *D2, *V; // matrices for SVD of S_Mat
    CvMat       *D, *A, *temp, *U_t; // matrices to reconstruct A_Mat
    CvMat       *H, *invH; // affine transform H, and its inverse

    m1      = cvCreateMat(3, 1, CV_32FC1);
    l1      = cvCreateMat(3, 1, CV_32FC1);
    m2      = cvCreateMat(3, 1, CV_32FC1);
    l2      = cvCreateMat(3, 1, CV_32FC1);

    S_Vec   = cvCreateMat(2, 1, CV_32FC1);
    S       = cvCreateMat(2, 2, CV_32FC1);
    U       = cvCreateMat(2, 2, CV_32FC1);
    D2      = cvCreateMat(2, 2, CV_32FC1);
    V       = cvCreateMat(2, 2, CV_32FC1);

    A       = cvCreateMat(2, 2, CV_32FC1);
    D       = cvCreateMat(2, 2, CV_32FC1);
    temp    = cvCreateMat(2, 2, CV_32FC1);
    U_t     = cvCreateMat(2, 2, CV_32FC1);

    H       = cvCreateMat(3, 3, CV_32FC1);
    invH    = cvCreateMat(3, 3, CV_32FC1);
```

```
// get the input image data
inHeight    = inImage->height;
inWidth     = inImage->width;
inStep      = inImage->widthStep;
inChannels  = inImage->nChannels;
inData      = (uchar *)inImage->imageData;

// create the affinely rectified image
outHeight   = inHeight;
outWidth    = inWidth;
outChannels = inChannels;
outStep     = outImage->widthStep;
outData     = (uchar *)outImage->imageData;

// initialize result image
for(i = 0; i < outHeight; i++){
    for(j = 0; j < outWidth; j++){
        for(k = 0; k < outChannels; k++){
            outData[i*outStep + j*outChannels + k] = 0;
        }
    }
}

// get 2 pairs of orthogonal lines to calculate the affine transform
cvCrossProduct(a, b, m1);
cvCrossProduct(b, d, l1);
cvCrossProduct(a, d, m2);
cvCrossProduct(b, c, l2);


// affine transform : H = [A t; 0 0 1]
// S = A * A^T
// calculate S (2 by 2)
// ---------------------------------------------------------------------
// (m1, l1) & (m2, l2) are the orthogonal line pairs in the affinely
// rectified image.
// (l1(1)m1(1), l1(1)m1(2) + l1(2)m1(1), l1(2)m1(2))s = 0
// (l2(1)m2(1), l2(1)m2(2) + l2(2)m2(1), l2(2)m2(2))s = 0
//  if we let s22 as 1, then above equations have the following form
//  [l1(1)m1(1) l1(2)m1(1)+l1(1)m1(2)] [s11] = [-l1(2)m1(2)]
//  [l2(1)m2(1) l2(2)m2(1)+l2(1)m2(2)] [s12]   [-l2(2)m2(2)]
float mat1, mat2, mat3, mat4, vec1, vec2;

mat1 = cvmGet(m1, 0, 0)*cvmGet(l1, 0, 0);
```

```
mat2 = cvmGet(m1, 1, 0)*cvmGet(l1, 0, 0)
       + cvmGet(m1, 0, 0)*cvmGet(l1, 1, 0);
mat3 = cvmGet(m2, 0, 0)*cvmGet(l2, 0, 0);
mat4 = cvmGet(m2, 1, 0)*cvmGet(l2, 0, 0)
       + cvmGet(m2, 0, 0)*cvmGet(l2, 1, 0);
vec1 = cvmGet(m1, 1, 0)*cvmGet(l1, 1, 0);
vec2 = cvmGet(m2, 1, 0)*cvmGet(l2, 1, 0);

float mat[4] = {mat1, mat2,
                mat3, mat4};
float vec[2] = {-vec1, -vec2};
Mat = cvMat(2, 2 , CV_32FC1, mat);
Vec = cvMat(2, 1 , CV_32FC1, vec);
cvSolve(&Mat, &Vec, S_Vec);

// S matrix and SVD
float s_mat[4] = {cvmGet(S_Vec, 0, 0), cvmGet(S_Vec, 1, 0),
                  cvmGet(S_Vec, 1, 0), 1}; // symmetric matrix
Array2CvMat(s_mat, S, 2, 2);
cvSVD(S, D2, U, V, CV_SVD_U_T|CV_SVD_V_T);
// S_Mat = U^T D2 V : opencv setting

// affine transform : A_Mat = U^T D V
cvPow(D2, D, 0.5);
cvTranspose(U, U_t);
cvMatMul(U_t, D, temp);
cvMatMul(temp, V, A);

// apply the transform to get the target image
// ---------------------------------------------------
// x' = H x
float h[9] = {cvmGet(A, 0, 0), cvmGet(A, 0, 1), shiftX,
              cvmGet(A, 1, 0), cvmGet(A, 1, 1), shiftY,
              0, 0, 1};
Array2CvMat(h, H, 3, 3);
cvInvert(H, invH);

if(IMPLEMENTATION == 1){ // case 1 : out(inv(H)x') = in(x')
    CvMat2Array(invH, h, 3, 3);
}else{ // case 2 : out(x) = in(Hx)
    CvMat2Array(H, h, 3, 3);
}

int ii, jj;
float x1, x2, x3;
```

```
    for(i = 0; i < inHeight-3; i++){ // case 1: i, j: x, ii, jj: x', x' = invHx
        for(j = 0; j < inWidth; j++){// case 2: i, j: x', ii, jj: x, x = Hx'
            for(k = 0; k < inChannels; k++){ // x: distorted, x': undistorted
                x1 = h[0] * j + h[1] * i + h[2];
                x2 = h[3] * j + h[4] * i + h[5];
                x3 = h[6] * j + h[7] * i + 1;
                ii = min(outHeight - 1, max(0, (int)(x2 / x3)));
                jj = min(outWidth - 1, max(0, (int)(x1 / x3)));
                if(IMPLEMENTATION == 1) // case 1 : out(inv(H)x') = in(x')
                    outData[ii*outStep + jj*outChannels + k]
                    = inData[i*inStep + j*inChannels + k];
                else // case 2 : out(x) = in(Hx)
                    outData[i*outStep + j*outChannels + k]
                    = inData[ii*inStep + jj*inChannels + k];
            }
        }
    }

    // create windows
    cvNamedWindow("input image", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("output image", CV_WINDOW_AUTOSIZE);

    // show the images
    cvShowImage("input image", inImage);
    cvShowImage("output image", outImage);
    // wait for a key
    cvWaitKey(0);
    cvDestroyWindow("input image");
    cvDestroyWindow("output image");

    // release matrices
    cvReleaseMat(&m1); cvReleaseMat(&l1); cvReleaseMat(&m2); cvReleaseMat(&l2);
    cvReleaseMat(&S_Vec); cvReleaseMat(&S);
    cvReleaseMat(&U); cvReleaseMat(&D2); cvReleaseMat(&V);
    cvReleaseMat(&D); cvReleaseMat(&A); cvReleaseMat(&U_t);
    cvReleaseMat(&temp);
    cvReleaseMat(&H); cvReleaseMat(&invH);

    // write output image
    sprintf(outImageName, "result2.jpg");
    if(!cvSaveImage(outImageName, outImage)){
        printf("Could not save: %s\n",outImageName);
    }
}
```

```
//
// function : CalculateVanishingLine
// usage : CalculateVanishingLine(a, b, c, d, vanishingLine);
// ---------------------------------------------------------------
// This function produces the vanishing line corresponding the give
// 4 points, a, b, c, and d.
//
void Rectification::CalculateVanishingLine(CvMat *a, CvMat *b,
                                           CvMat *c, CvMat *d,
                                           CvMat *vanishLine)
{

    CvMat *line_ab, *line_cd, *line_ac, *line_bd;
    CvMat *vanishPt1, *vanishPt2;

    line_ab = cvCreateMat(3, 1, CV_32FC1);
    line_cd = cvCreateMat(3, 1, CV_32FC1);
    line_ac = cvCreateMat(3, 1, CV_32FC1);
    line_bd = cvCreateMat(3, 1, CV_32FC1);

    vanishPt1 = cvCreateMat(3, 1, CV_32FC1);
    vanishPt2 = cvCreateMat(3, 1, CV_32FC1);

    // get 4 lines to calculate the 2 vanishing points
    cvCrossProduct(a, b, line_ab); // the line that passes the pt a & b
    cvCrossProduct(c, d, line_cd);
    cvCrossProduct(a, c, line_ac);
    cvCrossProduct(b, d, line_bd);

    // get vanishing points
    cvCrossProduct(line_ab, line_cd, vanishPt1);
    cvCrossProduct(line_ac, line_bd, vanishPt2);

    // get vanishing line
    cvCrossProduct(vanishPt1, vanishPt2, vanishLine);
}


void Rectification::Array2CvMat(float *arr, CvMat *cvArr, int row,
int column) {
    int i, j;

    for(i = 0; i < row; i++){
        for(j = 0; j < column; j++){
```

```
            cvmSet(cvArr, i, j, arr[i*column + j]);
        }
    }
}


void Rectification::CvMat2Array(CvMat *cvArr, float *arr, int row,
int column) {
    int i, j;

    for(i = 0; i < row; i++){
        for(j = 0; j < column; j++){
            arr[i*column + j] = cvmGet(cvArr, i, j);
        }
    }
}
```