# HW2: 2D Homography

J. Zapf[†]

## I. Problem Statement

The objective of this homework is to eliminate the projective distortion in an image of a planar scene using two differnt methods.

1) A two-step method. First the projective distortion is removed using two sets of parallel lines. Once the projective distortion is removed, only the affine distortion remains. The affine distortion is removed using two sets of orthogonal lines.
2) A single-step method. Five sets of orthogonal lines are used to determine the dual conic $C'^{*}_{\infty}$. The homography is then determined via a decomposition of $C'^{*}_{\infty}$.

## II. Solution

### A. 2-Step Method

In the 2-step method, we first remove the projective distortion from the image by using two sets of parallel lines. In all the example images, there exists several rectangular shapes. Thus we compute two sets of parallel lines from the points of a single rectangle. In the physical world, parallel lines intersect at the line at infinity, $l_\infty$. In the image space, however, parallel lines intersect at finite points due to the projective deformation. The line joining the points of intersection of two sets of parallel lines is called the vanishing line, $l_v = (l_1, l_2, l_3)$. The image is corrected for projective distortion via the homography that sends the vanishing line $l_v$ back to $l_\infty$. This homography is given by

$$H_P^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}. \tag{1}$$

[†] School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907

Once the projective distortion has been removed, we must then remove the affine distortion. To this end, we note that the angle $\theta$ between two lines $l$ and $m$ in the undistorted image can be expressed in terms of the projected lines $l'$ and $m'$ as

$$\cos\theta \propto l'^{\top} H_A C_\infty^* H_A^{\top} m' \tag{2}$$

$$= l'^{\top} C_\infty'^* m', \tag{3}$$

where $H_A$ represents an affine homogeneous transformation of the form

$$H_A = \begin{bmatrix} A & t \\ 0^{\top} & 1 \end{bmatrix}. \tag{4}$$

Note in particular that if $\theta = \pi/2$,

$$l'^{\top} C_\infty'^* m' = 0. \tag{5}$$

Now the dual conic $C_\infty'^*$ has the form

$$C_\infty'^* = \begin{bmatrix} S & 0 \\ 0^{\top} & 0 \end{bmatrix}, \tag{6}$$

where $S = KK^{\top}$. The orthogonality condition thus reduces to

$$(l'_1 m'_1, l'_1 m'_2 + l'_2 m'_1, l'_2 m'_2)s = 0, \tag{7}$$

where $s = (s_{11}, s_{12}, s_{22})^{\top}$. The matrix $S$ is symmetric and homogeneous. Thus it has two degrees of freedom and two orthogonal line pairs can be used to compute the null vector $s$. After determining $S$, an SVD is used to determine $K$ and subsequently $H_A$. Finally, the homography used to eliminate both projective and affine distortion is given by $H = H_P H_A$.

### *B. 2-Step Method*

It is possible to use the dual conic $C_\infty'^*$ to eliminate both projective and affine distortion. In this case, the homogeneous transformation takes the general form

$$H = \begin{bmatrix} K & 0 \\ v^{\top} & 1 \end{bmatrix}. \tag{8}$$

Thus, the dual conic $C'^{*}_{\infty}$ takes the form

$$C'^{*}_{\infty} = HC^{*}_{\infty}H^{\top} = \begin{bmatrix} KK^{\top} & Kv \\ v^{\top}K^{\top} & vv^{\top} \end{bmatrix}. \tag{9}$$

Furthermore, the orthogonality condition becomes

$$(l_1m_1, (l_1m_2 + l_2m_1)/2, l_2m_2, (l_1m_3 + l_3m_1)/2, (l_2m_3 + l_3m_2)/2, l_3m_3)\, c = 0, \tag{10}$$

where $s = (a, b, c, d, e, f)^{\top}$ are the parameters of the dual conic $C^{*}_{\infty}$. Stacking five such constraints from five sets of orthogonal lines, we compute the null vector $c$. To solve for the homography $H$, we first perform an SVD to determine $K$ as we did in the previous method. Once $K$ is determined, we can then solve the null-space equation $Kv = 0$. Finally, we compute the homography $H$ from $K$ and $v$.

### *C. Removing Distortion*

The pixel array size for most of the images used in this homework was $600 \times 800$. Thus, the bounding corners of each of these images, as defined in the image coordinate system, are

$$\left\{ \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 799 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 799 \\ 599 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 599 \\ 1 \end{bmatrix} \right\}. \tag{11}$$

To compute the corresponding bounds in the undistorted image, we simply multiply by the inverse homography, i.e. $H^{-1}$. Once these bounds are found, we define a grid on the undistorted image. Finally, to determine the pixel intensities of the corrected image, we multiply each of the grid coordinates in the undistorted image by the $H$ matrix to determine their corresponding pixel locations in the distorted image. A simple correspondence is thus established between grid locations in the undistorted image and pixel intensities in the original image.

## III. 2-Step Method Results


Fig. 1. adams01 original


Fig. 2. adams01 projective correction


Fig. 3. adams01 affine correction


Fig. 4. adams02 original

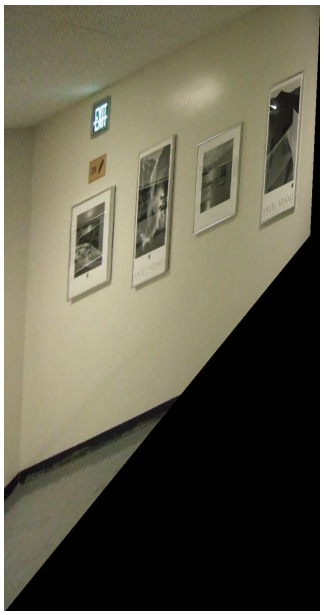


Fig. 5. adams02 projective correction


Fig. 6. adams02 affine correction

Fig. 7. board01 original

Fig. 8. board01 projective correction

Fig. 9. board01 affine correction


Fig. 10. door01 original

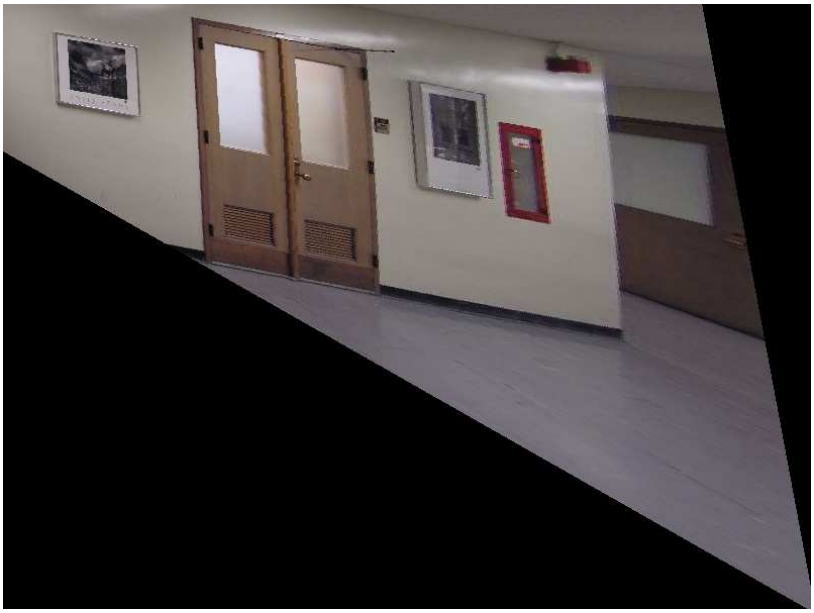

Fig. 11. door01 projective correction

Fig. 12. door01 affine correction

Fig. 13. book original

Fig. 14. book projective correction

Fig. 15. book affine correction

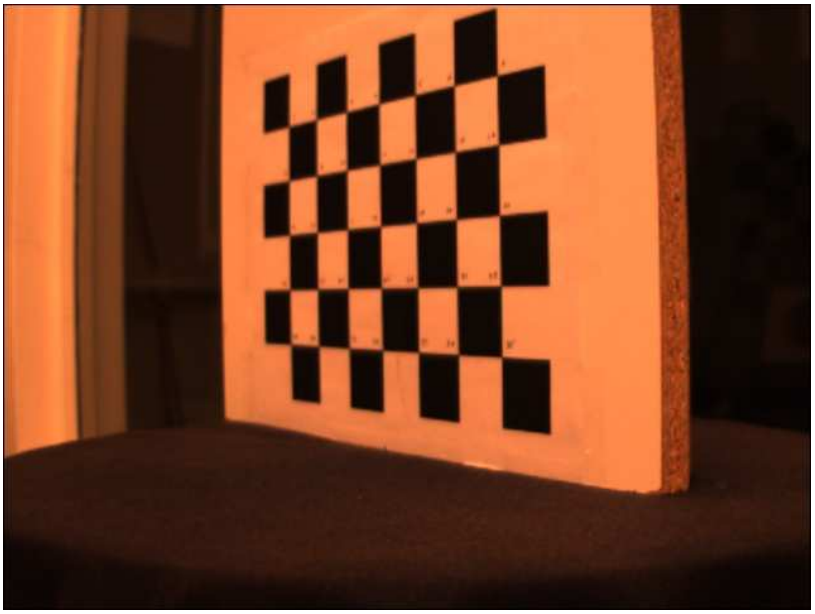

Fig. 16. calib original

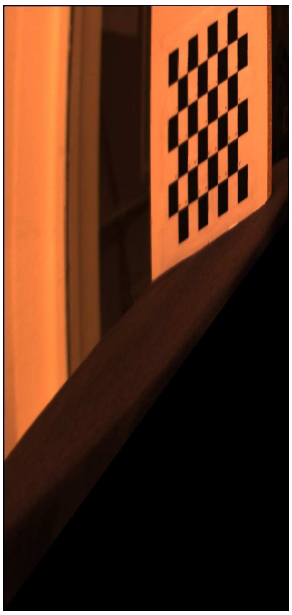

Fig. 17. calib projective correction

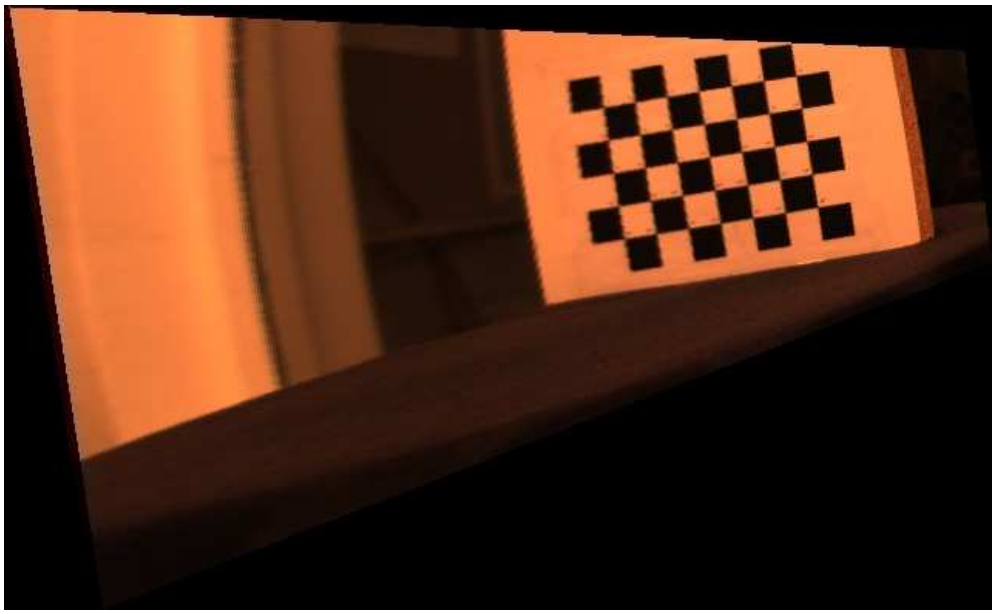

Fig. 18. calib affine correction

# IV. 1-Step Method Results


Fig. 19. adams01 original


Fig. 20. adams01 corrected


Fig. 21. adams02 original

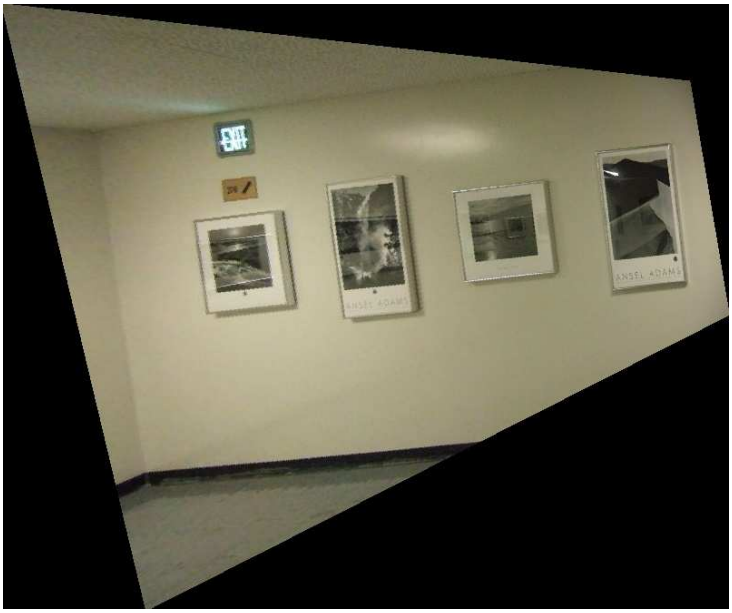


Fig. 22. adams02 corrected

Fig. 23. board01 original

Fig. 24. board01 corrected


Fig. 25. door01 original

Fig. 26. door01 corrected

Fig. 27. book original

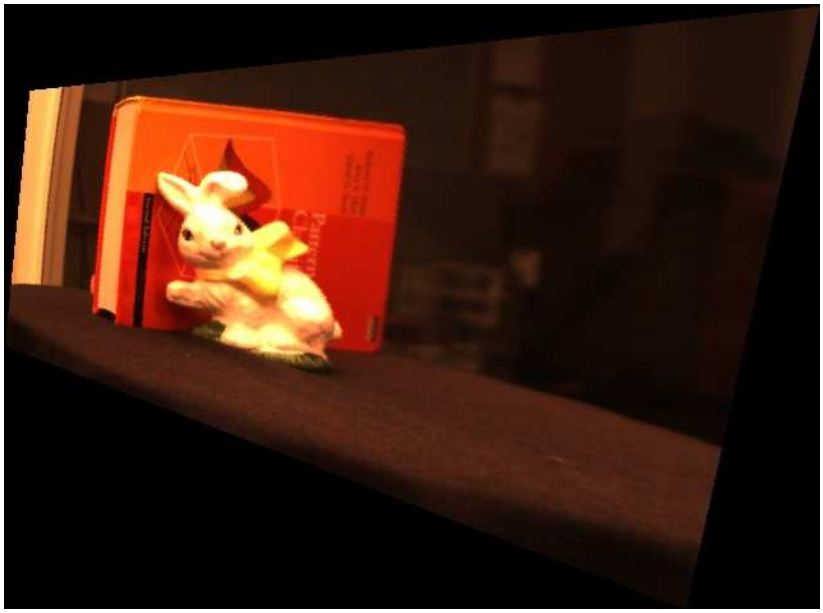

Fig. 28. book corrected

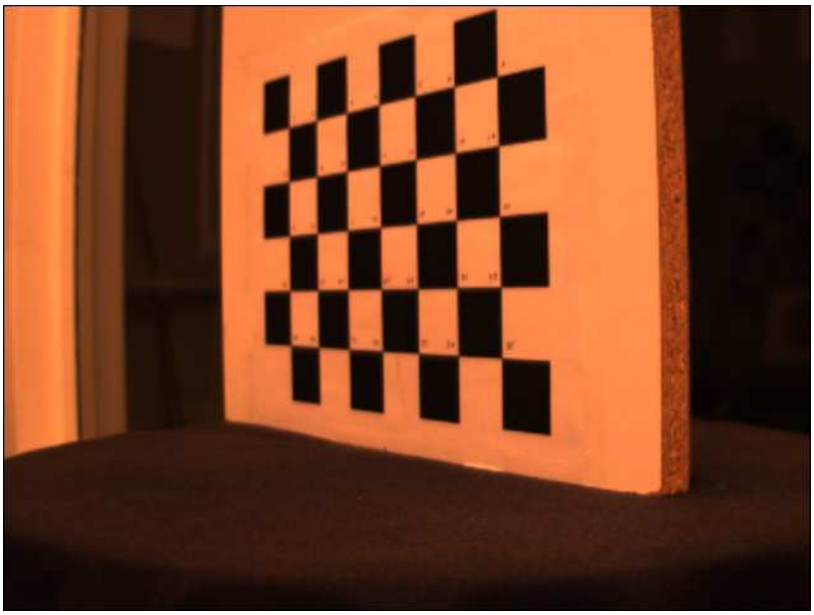

Fig. 29. calib original

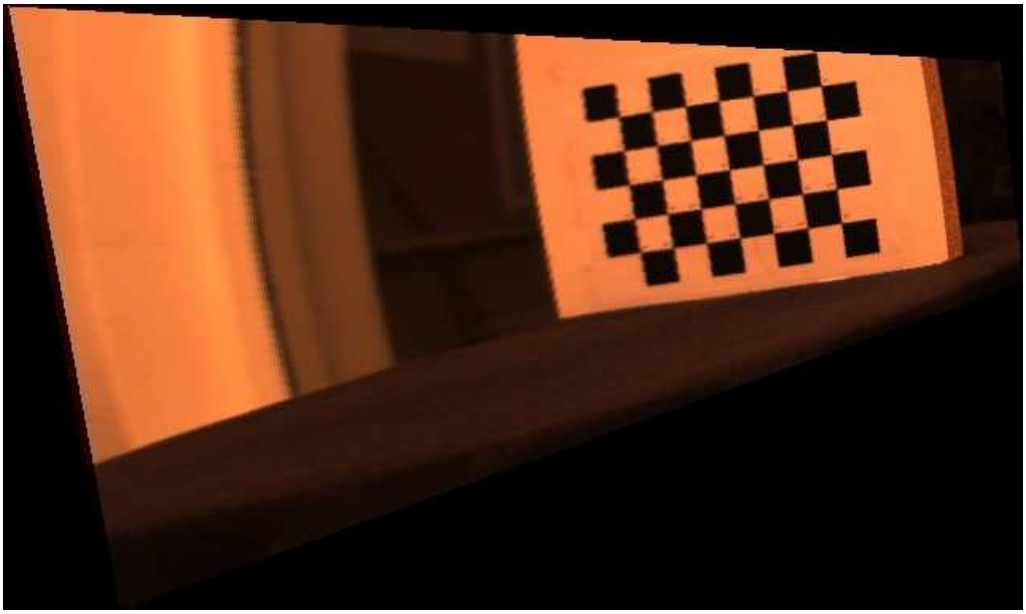

Fig. 30. calib corrected

# V. C Code for 2-Step Method

```
/*
 * main.c
 *
 *  Created on: Sep 20, 2010
 *      Author: jjzapf
 */

#include <opencv/cv.h>
#include <opencv/highgui.h>

//--------------------------------------------------------------------------
//  structure for storing image data
//--------------------------------------------------------------------------
struct frame {
    IplImage*   image;
    CvMat*      data;
    int         nPts;
};

//--------------------------------------------------------------------------
//  mouse event callback function
//--------------------------------------------------------------------------
void mouse_callback(int event, int x, int y, int flags, void* param)
{
    struct frame*   myPic = (struct frame*) param;
    static int      ptCnt = 0;
    int             x2, y2;
    CvPoint         pointA;
    CvPoint         pointB;

    switch(event)
    {
    case CV_EVENT_LBUTTONDOWN:
    {
        if (ptCnt < myPic->nPts) {
            pointA = cvPoint(x,y);
            cvCircle(myPic->image, pointA, 5, CV_RGB(255,0,0), 1, 8, 0);
            if (ptCnt > 0 && ptCnt < 4) {
                x2 = (int)(cvmGet(myPic->data, 0, ptCnt-1));
                y2 = (int)(cvmGet(myPic->data, 1, ptCnt-1));
                pointB = cvPoint(x2,y2);
                cvLine(myPic->image , pointA , pointB, CV_RGB(0,255,0), 5,
                        CV_AA, 0);
            }
            if (ptCnt == 3) {
                x2 = (int)(cvmGet(myPic->data, 0, 0));
                y2 = (int)(cvmGet(myPic->data, 1, 0));
                pointB = cvPoint(x2,y2);
                cvLine(myPic->image , pointA , pointB, CV_RGB(0,255,0), 5,
                        CV_AA, 0);
```

```
            }
            if (ptCnt > 4 && ptCnt < 6) {
                x2 = (int)(cvmGet(myPic->data, 0, ptCnt-1));
                y2 = (int)(cvmGet(myPic->data, 1, ptCnt-1));
                pointB = cvPoint(x2,y2);
                cvLine(myPic->image , pointA , pointB, CV_RGB(0,255,0), 5,
                        CV_AA, 0);
            }
            if (ptCnt > 6) {
                x2 = (int)(cvmGet(myPic->data, 0, ptCnt-1));
                y2 = (int)(cvmGet(myPic->data, 1, ptCnt-1));
                pointB = cvPoint(x2,y2);
                cvLine(myPic->image , pointA , pointB, CV_RGB(0,255,0), 5,
                        CV_AA, 0);
            }
            cvShowImage ("distorted image", myPic->image);
            cvmSet(myPic->data, 0, ptCnt, x);
            cvmSet(myPic->data, 1, ptCnt, y);
            ptCnt = ptCnt+1;
        }
    }
    }
}

//-------------------------------------------------------------------------
//  prints a matrix
//-------------------------------------------------------------------------
void PrintMat(CvMat* M, int rows, int cols)
{
    int i, j;

    for(i = 0; i < rows; i++) {
        printf("( ");
        for (j = 0; j < cols; j++) {
            printf("%15.4f", cvmGet(M,i,j));
        }
        printf(" )\n");
    }
}

//-------------------------------------------------------------------------
//  computes the points of intersection with the infinite line
//-------------------------------------------------------------------------
void line_intersect(int nLine ,CvMat* LINES, CvMat* pt)
{
    CvMat* ln1 = cvCreateMat(3, 1, CV_64F);
    CvMat* ln2 = cvCreateMat(3, 1, CV_64F);

    switch (nLine)
    {
    case 0:
    {
        cvmSet(ln1, 0, 0, cvmGet(LINES,0,3));
```

```
        cvmSet(ln1, 1, 0, cvmGet(LINES,1,3));
        cvmSet(ln1, 2, 0, cvmGet(LINES,2,3));
        cvmSet(ln2, 0, 0, cvmGet(LINES,0,1));
        cvmSet(ln2, 1, 0, cvmGet(LINES,1,1));
        cvmSet(ln2, 2, 0, cvmGet(LINES,2,1));
        cvCrossProduct(ln1, ln2, pt);
        break;
    }
    case 1:
    {
        cvmSet(ln1, 0, 0, cvmGet(LINES,0,0));
        cvmSet(ln1, 1, 0, cvmGet(LINES,1,0));
        cvmSet(ln1, 2, 0, cvmGet(LINES,2,0));
        cvmSet(ln2, 0, 0, cvmGet(LINES,0,2));
        cvmSet(ln2, 1, 0, cvmGet(LINES,1,2));
        cvmSet(ln2, 2, 0, cvmGet(LINES,2,2));
        cvCrossProduct(ln1, ln2, pt);
        break;
    }
    case 2:
    {
        cvmSet(ln1, 0, 0, cvmGet(LINES,0,1));
        cvmSet(ln1, 1, 0, cvmGet(LINES,1,1));
        cvmSet(ln1, 2, 0, cvmGet(LINES,2,1));
        cvmSet(ln2, 0, 0, cvmGet(LINES,0,3));
        cvmSet(ln2, 1, 0, cvmGet(LINES,1,3));
        cvmSet(ln2, 2, 0, cvmGet(LINES,2,3));
        cvCrossProduct(ln1, ln2, pt);
        break;
    }
    case 3:
    {
        cvmSet(ln1, 0, 0, cvmGet(LINES,0,2));
        cvmSet(ln1, 1, 0, cvmGet(LINES,1,2));
        cvmSet(ln1, 2, 0, cvmGet(LINES,2,2));
        cvmSet(ln2, 0, 0, cvmGet(LINES,0,0));
        cvmSet(ln2, 1, 0, cvmGet(LINES,1,0));
        cvmSet(ln2, 2, 0, cvmGet(LINES,2,0));
        cvCrossProduct(ln1, ln2, pt);
        break;
    }
    }
}

//-------------------------------------------------------------------------
//  computes the bounds in the undistorted coordinate system
//-------------------------------------------------------------------------
void compute_bounds(CvMat* Hinv, int height, int width, CvMat* bounds)
{
    CvMat* iCorners = cvCreateMat(3, 4, CV_64F);
    CvMat* wCorners = cvCreateMat(3, 4, CV_64F);

    cvmSet(iCorners, 0, 0, 0);
```

```
    cvmSet(iCorners, 1, 0, 0);
    cvmSet(iCorners, 2, 0, 1);

    cvmSet(iCorners, 0, 1, width-1);
    cvmSet(iCorners, 1, 1, 0);
    cvmSet(iCorners, 2, 1, 1);

    cvmSet(iCorners, 0, 2, width-1);
    cvmSet(iCorners, 1, 2, height-1);
    cvmSet(iCorners, 2, 2, 1);

    cvmSet(iCorners, 0, 3, 0);
    cvmSet(iCorners, 1, 3, height-1);
    cvmSet(iCorners, 2, 3, 1);

    cvMatMul(Hinv, iCorners, wCorners);

    double xMinA = cvmGet(wCorners,0,0)/cvmGet(wCorners,2,0);
    double xMinB = cvmGet(wCorners,0,3)/cvmGet(wCorners,2,3);
    double xMaxA = cvmGet(wCorners,0,1)/cvmGet(wCorners,2,1);
    double xMaxB = cvmGet(wCorners,0,2)/cvmGet(wCorners,2,2);

    double yMinA = cvmGet(wCorners,1,0)/cvmGet(wCorners,2,0);
    double yMinB = cvmGet(wCorners,1,1)/cvmGet(wCorners,2,1);
    double yMaxA = cvmGet(wCorners,1,2)/cvmGet(wCorners,2,2);
    double yMaxB = cvmGet(wCorners,1,3)/cvmGet(wCorners,2,3);

    double xMin = xMinA;
    double xMax = xMaxA;
    if (xMinB < xMin)
        xMin = xMinB;
    if (xMaxB > xMax)
        xMax = xMaxB;

    double yMin = yMinA;
    double yMax = yMaxA;
    if (yMinB < yMin)
        yMin = yMinB;
    if (yMaxB > yMax)
        yMax = yMaxB;

    cvmSet(bounds,0,0,xMin);
    cvmSet(bounds,0,1,xMax);
    cvmSet(bounds,0,2,yMin);
    cvmSet(bounds,0,3,yMax);
}

//--------------------------------------------------------------------------
//  main
//--------------------------------------------------------------------------
int main(int argc, char *argv[])
{
    int             i, j, flag;
```

```
int             nPts = 8;
char            inFile[256];
char            outFileA[256];
char            outFileB[256];
double          temp, tmp2;
struct frame    myPic = {0, cvCreateMat(2,nPts,CV_64F), nPts};

/*
sprintf(inFile, "images/adams01.jpg");
sprintf(outFileA, "images/adams01P.jpg");
sprintf(outFileB, "images/adams01A.jpg");
*/

/*
sprintf(inFile, "images/adams02.jpg");
sprintf(outFileA, "images/adams02P.jpg");
sprintf(outFileB, "images/adams02A.jpg");
*/

/*
sprintf(inFile, "images/door01.jpg");
sprintf(outFileA, "images/door01P.jpg");
sprintf(outFileB, "images/door01A.jpg");
*/

/*
sprintf(inFile, "images/board01.jpg");
sprintf(outFileA, "images/board01P.jpg");
sprintf(outFileB, "images/board01A.jpg");
*/

/*
sprintf(inFile, "images/book.jpg");
sprintf(outFileA, "images/bookP.jpg");
sprintf(outFileB, "images/bookA.jpg");
*/

sprintf(inFile, "images/calib.jpg");
sprintf(outFileA, "images/calibP.jpg");
sprintf(outFileB, "images/calibA.jpg");

//-------------------------------------------------------------------------
//  extracts points from the image
//-------------------------------------------------------------------------
myPic.image = cvLoadImage(inFile, CV_LOAD_IMAGE_ANYDEPTH |
        CV_LOAD_IMAGE_ANYCOLOR);
if (myPic.image == 0) {
    printf("Cannot load file %s.\n", inFile);
}
cvNamedWindow("distorted image", CV_WINDOW_AUTOSIZE);
cvSetMouseCallback("distorted image", mouse_callback, (void*) &myPic);
cvShowImage("distorted image", myPic.image);
cvWaitKey(0);
```

```
cvDestroyWindow("distorted image");
cvReleaseImage(&myPic.image);

IplImage* image = cvLoadImage(inFile, CV_LOAD_IMAGE_ANYDEPTH |
        CV_LOAD_IMAGE_ANYCOLOR);
if (image == 0) {
    printf("Cannot load file %s.\n", inFile);
}

//--------------------------------------------------------------------
//  creates the point and line matrices
//--------------------------------------------------------------------
CvMat* pt1 = cvCreateMat(3, 1, CV_64F);
CvMat* pt2 = cvCreateMat(3, 1, CV_64F);
CvMat* line = cvCreateMat(3, 1, CV_64F);
CvMat* dist = cvCreateMat(1, 4, CV_64F);
CvMat* LINES = cvCreateMat(3, 6, CV_64F);

//--------------------------------------------------------------------
//  computes the lines of the rectangle
//--------------------------------------------------------------------
cvmSet(pt1, 2, 0, 1.0);
cvmSet(pt2, 2, 0, 1.0);
for (j = 0; j < 4; j++) {
    if (j == 3) {
        cvmSet(pt1, 0, 0, cvmGet(myPic.data,0,3));
        cvmSet(pt1, 1, 0, cvmGet(myPic.data,1,3));
        cvmSet(pt2, 0, 0, cvmGet(myPic.data,0,0));
        cvmSet(pt2, 1, 0, cvmGet(myPic.data,1,0));
    }
    else
    {
        cvmSet(pt1, 0, 0, cvmGet(myPic.data,0,j));
        cvmSet(pt1, 1, 0, cvmGet(myPic.data,1,j));
        cvmSet(pt2, 0, 0, cvmGet(myPic.data,0,j+1));
        cvmSet(pt2, 1, 0, cvmGet(myPic.data,1,j+1));
    }
    cvCrossProduct(pt1, pt2, line);
    temp = 0.0;
    for (i = 0; i < 3; i++) {
        cvmSet(LINES, i, j, cvmGet(line,i,0));
        temp = temp+pow(cvmGet(pt2,i,0)-cvmGet(pt1,i,0),2);
    }
    cvmSet(dist, 0, j, temp);
}

//--------------------------------------------------------------------
//  computes the lines of the X
//--------------------------------------------------------------------
cvmSet(pt1, 0, 0, cvmGet(myPic.data,0,4));
cvmSet(pt1, 1, 0, cvmGet(myPic.data,1,4));
cvmSet(pt2, 0, 0, cvmGet(myPic.data,0,5));
cvmSet(pt2, 1, 0, cvmGet(myPic.data,1,5));
```

```
cvCrossProduct(pt1, pt2, line);
for (i = 0; i < 3; i++) {
    cvmSet(LINES, i, 4, cvmGet(line,i,0));
}

cvmSet(pt1, 0, 0, cvmGet(myPic.data,0,6));
cvmSet(pt1, 1, 0, cvmGet(myPic.data,1,6));
cvmSet(pt2, 0, 0, cvmGet(myPic.data,0,7));
cvmSet(pt2, 1, 0, cvmGet(myPic.data,1,7));
cvCrossProduct(pt1, pt2, line);
for (i = 0; i < 3; i++) {
    cvmSet(LINES, i, 5, cvmGet(line,i,0));
}

//-------------------------------------------------------------------------
//  finds the shortest line in the rectangle
//-------------------------------------------------------------------------
int n1 = 0;
int n2 = 0;

temp = cvmGet(dist, 0, 0);
for (j = 1; j < 4; j++) {
    if (cvmGet(dist, 0, j) < temp) {
        temp = cvmGet(dist, 0, j);
        n1 = j;
    }
}

switch (n1)
{
case 0:
{
    if (cvmGet(dist,0,1) < cvmGet(dist,0,3))
        n2 = 1;
    else
        n2 = 3;
    break;
}
case 1:
{
    if (cvmGet(dist,0,2) < cvmGet(dist,0,0))
        n2 = 2;
    else
        n2 = 0;
    break;
}
case 2:
{
    if (cvmGet(dist,0,3) < cvmGet(dist,0,1))
        n2 = 3;
    else
        n2 = 1;
    break;
```

```
}
case 3:
{
    if (cvmGet(dist,0,0) < cvmGet(dist,0,2))
        n2 = 0;
    else
        n2 = 2;
    break;
}
}

//-----------------------------------------------------------------------
//  computes the points of intersection with the infinite line
//-----------------------------------------------------------------------
line_intersect(n1, LINES, pt1);
line_intersect(n2, LINES, pt2);

//-----------------------------------------------------------------------
//  computes the vanishing line
//-----------------------------------------------------------------------
cvCrossProduct(pt1, pt2, line);
cvmSet(line,0,0,cvmGet(line,0,0)/cvmGet(line,2,0));
cvmSet(line,1,0,cvmGet(line,1,0)/cvmGet(line,2,0));
cvmSet(line,2,0,1.0);

//-----------------------------------------------------------------------
//  computes the homography for removing the projective distortion
//-----------------------------------------------------------------------
CvMat* HP = cvCreateMat(3, 3, CV_64F);
CvMat* HPinv = cvCreateMat(3, 3, CV_64F);

cvZero(HPinv);
cvmSet(HPinv, 0, 0, 1.0);
cvmSet(HPinv, 1, 1, 1.0);
cvmSet(HPinv, 2, 0, cvmGet(line, 0, 0));
cvmSet(HPinv, 2, 1, cvmGet(line, 1, 0));
cvmSet(HPinv, 2, 2, cvmGet(line, 2, 0));

cvZero(HP);
cvmSet(HP, 0, 0, 1.0);
cvmSet(HP, 1, 1, 1.0);
cvmSet(HP, 2, 0, -cvmGet(line,0,0)/cvmGet(line,2,0));
cvmSet(HP, 2, 1, -cvmGet(line,1,0)/cvmGet(line,2,0));
cvmSet(HP, 2, 2, 1.0/cvmGet(line,2,0));

cvNormalize(HP, HP, 1, 0, CV_L2, NULL);
cvNormalize(HPinv, HPinv, 1, 0, CV_L2, NULL);

for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
        cvmSet(HPinv, i, j, cvmGet(HPinv,i,j)/cvmGet(HPinv,2,2));
        cvmSet(HP, i, j, cvmGet(HP,i,j)/cvmGet(HP,2,2));
    }
```

```
}

//-----------------------------------------------------------------------
//  removes projective distortion from the lines
//-----------------------------------------------------------------------
CvMat* HPT = cvCreateMat(3, 3, CV_64F);
cvTranspose(HP,HPT);
cvMatMul(HPT,LINES,LINES);

//-----------------------------------------------------------------------
//  defines the A matrix
//-----------------------------------------------------------------------
CvMat* A = cvCreateMat(2, 3, CV_64F);

double l1, l2, l3;
double m1, m2, m3;

l1 = cvmGet(LINES,0,0)/cvmGet(LINES,2,0);
l2 = cvmGet(LINES,1,0)/cvmGet(LINES,2,0);
m1 = cvmGet(LINES,0,1)/cvmGet(LINES,2,1);
m2 = cvmGet(LINES,1,1)/cvmGet(LINES,2,1);

cvmSet(A, 0, 0, l1*m1);
cvmSet(A, 0, 1, l1*m2+l2*m1);
cvmSet(A, 0, 2, l2*m2);

l1 = cvmGet(LINES,0,4)/cvmGet(LINES,2,4);
l2 = cvmGet(LINES,1,4)/cvmGet(LINES,2,4);
m1 = cvmGet(LINES,0,5)/cvmGet(LINES,2,5);
m2 = cvmGet(LINES,1,5)/cvmGet(LINES,2,5);

cvmSet(A, 1, 0, l1*m1);
cvmSet(A, 1, 1, l1*m2+l2*m1);
cvmSet(A, 1, 2, l2*m2);

//-----------------------------------------------------------------------
//  computes the S matrix
//-----------------------------------------------------------------------
CvMat* V = cvCreateMat(3, 3, CV_64F);
CvMat* W = cvCreateMat(2, 3, CV_64F);
cvSVD(A, W, NULL, V, 0);
CvMat* S = cvCreateMat(2, 2, CV_64F);
cvmSet(S, 0, 0, cvmGet(V,0,2));
cvmSet(S, 0, 1, cvmGet(V,1,2));
cvmSet(S, 1, 0, cvmGet(V,1,2));
cvmSet(S, 1, 1, cvmGet(V,2,2));

//-----------------------------------------------------------------------
//  computes the K matrix
//-----------------------------------------------------------------------
CvMat* U = cvCreateMat(2, 2, CV_64F);
CvMat* D2 = cvCreateMat(2, 2, CV_64F);
CvMat* D = cvCreateMat(2, 2, CV_64F);
```

```
CvMat* UD = cvCreateMat(2, 2, CV_64F);
CvMat* K = cvCreateMat(2, 2, CV_64F);

cvSVD(S, D2, U, NULL, 0);
cvPow(D2, D, 0.5);
cvMatMul(U, D, UD);
cvGEMM(UD, U, 1.0, NULL, 0, K, CV_GEMM_B_T);

//---------------------------------------------------------------------
//  computes the homography for removing the affine distortion
//---------------------------------------------------------------------
CvMat* HA = cvCreateMat(3, 3, CV_64F);
CvMat* HAinv = cvCreateMat(3, 3, CV_64F);
cvZero(HA);
for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; j++) {
        cvmSet(HA, i, j, cvmGet(K, i, j));
    }
}
cvmSet(HA, 2, 2, 1.0);
cvNormalize(HA, HA, 1, 0, CV_L2, NULL);
cvInvert(HA, HAinv, CV_LU);
cvNormalize(HAinv, HAinv, 1, 0, CV_L2, NULL);

for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
        cvmSet(HAinv, i, j, cvmGet(HAinv,i,j)/cvmGet(HAinv,2,2));
        cvmSet(HA, i, j, cvmGet(HA,i,j)/cvmGet(HA,2,2));
    }
}

//---------------------------------------------------------------------
//  computes the homography for removing all the distortion
//---------------------------------------------------------------------
CvMat* H = cvCreateMat(3, 3, CV_64F);
CvMat* Hinv = cvCreateMat(3, 3, CV_64F);
cvMatMul(HP, HA, H);
cvMatMul(HAinv, HPinv, Hinv);

//---------------------------------------------------------------------
//  computes the bounds in the undistorted coordinate system
//---------------------------------------------------------------------
CvMat* bounds = cvCreateMat(1, 4, CV_64F);
compute_bounds(HPinv, image->height, image->width, bounds);
double xMin = cvmGet(bounds,0,0);
double xMax = cvmGet(bounds,0,1);
double yMin = cvmGet(bounds,0,2);
double yMax = cvmGet(bounds,0,3);

double scale_factor = ((double)(image->width))/(xMax-xMin);
int new_height = (int)((yMax-yMin)*(scale_factor));

//---------------------------------------------------------------------
```

```
//  removes projective distortion from the image
//---------------------------------------------------------------------
IplImage *pImage = cvCreateImage(cvSize(image->width, new_height),
        IPL_DEPTH_8U, 3);
cvZero(pImage);
CvMat* Pw = cvCreateMat(3, 1, CV_64F);
CvMat* Pi = cvCreateMat(3, 1, CV_64F);
CvScalar pixel;

double step = 1.0/scale_factor;
int xi, yi;

cvmSet(Pw, 2, 0, 1.0);
for (i = 0; i < pImage->height; i++) {
    cvmSet(Pw, 1, 0, ((double)(i))*step+yMin);
    for (j = 0; j < pImage->width; j++) {
        cvmSet(Pw, 0, 0, ((double)(j))*step+xMin);
        cvMatMul(HP, Pw, Pi);
        xi = ((int)(cvmGet(Pi,0,0)/cvmGet(Pi,2,0)));
        yi = ((int)(cvmGet(Pi,1,0)/cvmGet(Pi,2,0)));
        if (xi < 0 || xi >= image->width || yi < 0 ||
                yi >= image->height)
        {
            continue;
        }
        pixel = cvGet2D(image,yi,xi);
        cvSet2D(pImage,i,j,pixel);
    }
}

cvSaveImage(outFileA, pImage, 0);
cvNamedWindow("projective correction", CV_WINDOW_AUTOSIZE);
cvShowImage("projective correction", pImage);
cvWaitKey(0);
cvDestroyWindow("projective correction");

//---------------------------------------------------------------------
//  computes the bounds in the undistorted coordinate system
//---------------------------------------------------------------------
compute_bounds(Hinv, image->height, image->width, bounds);
xMin = cvmGet(bounds,0,0);
xMax = cvmGet(bounds,0,1);
yMin = cvmGet(bounds,0,2);
yMax = cvmGet(bounds,0,3);

scale_factor = ((double)(image->width))/(xMax-xMin);
new_height = (int)((yMax-yMin)*(scale_factor));

//---------------------------------------------------------------------
//  removes all the distortion from the image
//---------------------------------------------------------------------
IplImage *aImage = cvCreateImage(cvSize(image->width, new_height),
        IPL_DEPTH_8U, 3);
```

```
    cvZero(aImage);

    step = 1.0/scale_factor;

    cvmSet(Pw, 2, 0, 1.0);
    for (i = 0; i < aImage->height; i++) {
        cvmSet(Pw, 1, 0, ((double)(i))*step+yMin);
        for (j = 0; j < aImage->width; j++) {
            cvmSet(Pw, 0, 0, ((double)(j))*step+xMin);
            cvMatMul(H, Pw, Pi);
            xi = ((int)(cvmGet(Pi,0,0)/cvmGet(Pi,2,0)));
            yi = ((int)(cvmGet(Pi,1,0)/cvmGet(Pi,2,0)));
            if (xi < 0 || xi >= image->width || yi < 0 ||
                    yi >= image->height)
            {
                continue;
            }
            pixel = cvGet2D(image,yi,xi);
            cvSet2D(aImage,i,j,pixel);
        }
    }

    cvSaveImage(outFileB, aImage, 0);
    cvNamedWindow("affine correction", CV_WINDOW_AUTOSIZE);
    cvShowImage("affine correction", aImage);
    cvWaitKey(0);
    cvDestroyWindow("affine correction");

    //-------------------------------------------------------------------------
    //  release the images
    //-------------------------------------------------------------------------
    cvReleaseImage(&image);
    cvReleaseImage(&pImage);
    cvReleaseImage(&aImage);

    return 0;
}
```

# VI. C Code for 1-Step Method

```
/*
 * main.c
 *
 *  Created on: Sep 21, 2010
 *      Author: jjzapf
 */

#include <stdio.h>
#include <opencv/cv.h>
#include <opencv/highgui.h>

//--------------------------------------------------------------------------
//  structure for storing image data
//--------------------------------------------------------------------------
struct frame {
    IplImage*   image;
    CvMat*      data;
    int         nPts;
};

//--------------------------------------------------------------------------
//  mouse event callback function
//--------------------------------------------------------------------------
void mouse_callback(int event, int x, int y, int flags, void* param)
{
    struct frame*   myPic = (struct frame*) param;
    static int      ptCnt = 0;
    int             x2, y2;
    CvPoint         pointA;
    CvPoint         pointB;

    switch(event)
    {
    case CV_EVENT_LBUTTONDOWN:
    {
        if (ptCnt < myPic->nPts) {
            pointA = cvPoint(x,y);
            cvCircle(myPic->image, pointA, 5, CV_RGB(255,0,0), 1, 8, 0);
            if (ptCnt > 0 && ptCnt < 4) {
                x2 = (int)(cvmGet(myPic->data, 0, ptCnt-1));
                y2 = (int)(cvmGet(myPic->data, 1, ptCnt-1));
                pointB = cvPoint(x2,y2);
                cvLine(myPic->image , pointA , pointB, CV_RGB(0,255,0), 5,
                        CV_AA, 0);
            }
            if (ptCnt == 3) {
                x2 = (int)(cvmGet(myPic->data, 0, 0));
                y2 = (int)(cvmGet(myPic->data, 1, 0));
                pointB = cvPoint(x2,y2);
                cvLine(myPic->image , pointA , pointB, CV_RGB(0,255,0), 5,
```

```
                        CV_AA, 0);
            }
            if (ptCnt > 4 && ptCnt < 6) {
                x2 = (int)(cvmGet(myPic->data, 0, ptCnt-1));
                y2 = (int)(cvmGet(myPic->data, 1, ptCnt-1));
                pointB = cvPoint(x2,y2);
                cvLine(myPic->image , pointA , pointB, CV_RGB(0,255,0), 5,
                        CV_AA, 0);
            }
            if (ptCnt > 6) {
                x2 = (int)(cvmGet(myPic->data, 0, ptCnt-1));
                y2 = (int)(cvmGet(myPic->data, 1, ptCnt-1));
                pointB = cvPoint(x2,y2);
                cvLine(myPic->image , pointA , pointB, CV_RGB(0,255,0), 5,
                        CV_AA, 0);
            }
            cvShowImage ("distorted image", myPic->image);
            cvmSet(myPic->data, 0, ptCnt, x);
            cvmSet(myPic->data, 1, ptCnt, y);
            ptCnt = ptCnt+1;
        }
    }
    }
}

//-------------------------------------------------------------------------
//  prints a matrix
//-------------------------------------------------------------------------
void PrintMat(CvMat* M, int rows, int cols)
{
    int i, j;

    for(i = 0; i < rows; i++) {
        printf("( ");
        for (j = 0; j < cols; j++) {
            printf("%15.4f", cvmGet(M,i,j));
        }
        printf(" )\n");
    }
}

//-------------------------------------------------------------------------
//  computes the bounds in the undistorted coordinate system
//-------------------------------------------------------------------------
void compute_bounds(CvMat* Hinv, int height, int width, CvMat* bounds)
{
    CvMat* iCorners = cvCreateMat(3, 4, CV_64F);
    CvMat* wCorners = cvCreateMat(3, 4, CV_64F);

    cvmSet(iCorners, 0, 0, 0);
    cvmSet(iCorners, 1, 0, 0);
    cvmSet(iCorners, 2, 0, 1);
```

```
    cvmSet(iCorners, 0, 1, width-1);
    cvmSet(iCorners, 1, 1, 0);
    cvmSet(iCorners, 2, 1, 1);

    cvmSet(iCorners, 0, 2, width-1);
    cvmSet(iCorners, 1, 2, height-1);
    cvmSet(iCorners, 2, 2, 1);

    cvmSet(iCorners, 0, 3, 0);
    cvmSet(iCorners, 1, 3, height-1);
    cvmSet(iCorners, 2, 3, 1);

    cvMatMul(Hinv, iCorners, wCorners);

    double xMinA = cvmGet(wCorners,0,0)/cvmGet(wCorners,2,0);
    double xMinB = cvmGet(wCorners,0,3)/cvmGet(wCorners,2,3);
    double xMaxA = cvmGet(wCorners,0,1)/cvmGet(wCorners,2,1);
    double xMaxB = cvmGet(wCorners,0,2)/cvmGet(wCorners,2,2);

    double yMinA = cvmGet(wCorners,1,0)/cvmGet(wCorners,2,0);
    double yMinB = cvmGet(wCorners,1,1)/cvmGet(wCorners,2,1);
    double yMaxA = cvmGet(wCorners,1,2)/cvmGet(wCorners,2,2);
    double yMaxB = cvmGet(wCorners,1,3)/cvmGet(wCorners,2,3);

    double xMin = xMinA;
    double xMax = xMaxA;
    if (xMinB < xMin)
        xMin = xMinB;
    if (xMaxB > xMax)
        xMax = xMaxB;

    double yMin = yMinA;
    double yMax = yMaxA;
    if (yMinB < yMin)
        yMin = yMinB;
    if (yMaxB > yMax)
        yMax = yMaxB;

    cvmSet(bounds,0,0,xMin);
    cvmSet(bounds,0,1,xMax);
    cvmSet(bounds,0,2,yMin);
    cvmSet(bounds,0,3,yMax);
}

//--------------------------------------------------------------------------
//  main
//--------------------------------------------------------------------------
int main(int argc, char *argv[])
{
    int             i, j;
    int             nPts = 8;
    char            inFile[256];
    char            outFile[256];
```

```
double          tmp1, tmp2;
struct frame    myPic = {0, cvCreateMat(2,nPts,CV_64F), nPts};

/*
sprintf(inFile, "images/adams01.jpg");
sprintf(outFile, "images/adams01new.jpg");
*/

sprintf(inFile, "images/adams02.jpg");
sprintf(outFile, "images/adams02new.jpg");

/*
sprintf(inFile, "images/door01.jpg");
sprintf(outFile, "images/door01new.jpg");
*/

/*
sprintf(inFile, "images/board01.jpg");
sprintf(outFile, "images/board01new.jpg");
*/

/*
sprintf(inFile, "images/book.jpg");
sprintf(outFile, "images/booknew.jpg");
*/

/*
sprintf(inFile, "images/calib.jpg");
sprintf(outFile, "images/calibnew.jpg");
*/

//-----------------------------------------------------------------------
//  extracts points from the image
//-----------------------------------------------------------------------
myPic.image = cvLoadImage(inFile, CV_LOAD_IMAGE_ANYDEPTH |
        CV_LOAD_IMAGE_ANYCOLOR);
if (myPic.image == 0) {
    printf("Cannot load file %s.\n", inFile);
}
cvNamedWindow("distorted image", CV_WINDOW_AUTOSIZE);
cvSetMouseCallback("distorted image", mouse_callback,
        (void*) &myPic);
cvShowImage("distorted image", myPic.image);
cvWaitKey(0);
cvDestroyWindow("distorted image");
cvReleaseImage(&myPic.image);

IplImage* image = cvLoadImage(inFile, CV_LOAD_IMAGE_ANYDEPTH |
        CV_LOAD_IMAGE_ANYCOLOR);
if (image == 0) {
    printf("Cannot load file %s.\n", inFile);
}
```

```
//-----------------------------------------------------------------------
//  creates the point and line matrices
//-----------------------------------------------------------------------
CvMat* pt1 = cvCreateMat(3, 1, CV_64F);
CvMat* pt2 = cvCreateMat(3, 1, CV_64F);
CvMat* line = cvCreateMat(3, 1, CV_64F);
CvMat* LINES = cvCreateMat(3, 6, CV_64F);

//-----------------------------------------------------------------------
//  computes the lines of the rectangle
//-----------------------------------------------------------------------
cvmSet(pt1, 2, 0, 1.0);
cvmSet(pt2, 2, 0, 1.0);
for (j = 0; j < 4; j++) {
    if (j == 3) {
        cvmSet(pt1, 0, 0, cvmGet(myPic.data,0,3));
        cvmSet(pt1, 1, 0, cvmGet(myPic.data,1,3));
        cvmSet(pt2, 0, 0, cvmGet(myPic.data,0,0));
        cvmSet(pt2, 1, 0, cvmGet(myPic.data,1,0));
    }
    else
    {
        cvmSet(pt1, 0, 0, cvmGet(myPic.data,0,j));
        cvmSet(pt1, 1, 0, cvmGet(myPic.data,1,j));
        cvmSet(pt2, 0, 0, cvmGet(myPic.data,0,j+1));
        cvmSet(pt2, 1, 0, cvmGet(myPic.data,1,j+1));
    }
    cvCrossProduct(pt1, pt2, line);
    for (i = 0; i < 3; i++) {
        cvmSet(LINES, i, j, cvmGet(line,i,0));
    }
}

//-----------------------------------------------------------------------
//  computes the lines of the X
//-----------------------------------------------------------------------
cvmSet(pt1, 0, 0, cvmGet(myPic.data,0,4));
cvmSet(pt1, 1, 0, cvmGet(myPic.data,1,4));
cvmSet(pt2, 0, 0, cvmGet(myPic.data,0,5));
cvmSet(pt2, 1, 0, cvmGet(myPic.data,1,5));
cvCrossProduct(pt1, pt2, line);
for (i = 0; i < 3; i++) {
    cvmSet(LINES, i, 4, cvmGet(line,i,0));
}

cvmSet(pt1, 0, 0, cvmGet(myPic.data,0,6));
cvmSet(pt1, 1, 0, cvmGet(myPic.data,1,6));
cvmSet(pt2, 0, 0, cvmGet(myPic.data,0,7));
cvmSet(pt2, 1, 0, cvmGet(myPic.data,1,7));
cvCrossProduct(pt1, pt2, line);
for (i = 0; i < 3; i++) {
    cvmSet(LINES, i, 5, cvmGet(line,i,0));
}
```

```
//-------------------------------------------------------------------------
//  defines the A matrix
//-------------------------------------------------------------------------
CvMat* A = cvCreateMat(5, 6, CV_64F);

double l1, l2, l3;
double m1, m2, m3;

l1 = cvmGet(LINES,0,0);
l2 = cvmGet(LINES,1,0);
l3 = cvmGet(LINES,2,0);
m1 = cvmGet(LINES,0,1);
m2 = cvmGet(LINES,1,1);
m3 = cvmGet(LINES,2,1);
cvmSet(A, 0, 0, l1*m1);
cvmSet(A, 0, 1, (l1*m2+l2*m1)/2);
cvmSet(A, 0, 2, l2*m2);
cvmSet(A, 0, 3, (l1*m3+l3*m1)/2);
cvmSet(A, 0, 4, (l2*m3+l3*m2)/2);
cvmSet(A, 0, 5, l3*m3);

l1 = cvmGet(LINES,0,1);
l2 = cvmGet(LINES,1,1);
l3 = cvmGet(LINES,2,1);
m1 = cvmGet(LINES,0,2);
m2 = cvmGet(LINES,1,2);
m3 = cvmGet(LINES,2,2);
cvmSet(A, 1, 0, l1*m1);
cvmSet(A, 1, 1, (l1*m2+l2*m1)/2);
cvmSet(A, 1, 2, l2*m2);
cvmSet(A, 1, 3, (l1*m3+l3*m1)/2);
cvmSet(A, 1, 4, (l2*m3+l3*m2)/2);
cvmSet(A, 1, 5, l3*m3);

l1 = cvmGet(LINES,0,2);
l2 = cvmGet(LINES,1,2);
l3 = cvmGet(LINES,2,2);
m1 = cvmGet(LINES,0,3);
m2 = cvmGet(LINES,1,3);
m3 = cvmGet(LINES,2,3);
cvmSet(A, 2, 0, l1*m1);
cvmSet(A, 2, 1, (l1*m2+l2*m1)/2);
cvmSet(A, 2, 2, l2*m2);
cvmSet(A, 2, 3, (l1*m3+l3*m1)/2);
cvmSet(A, 2, 4, (l2*m3+l3*m2)/2);
cvmSet(A, 2, 5, l3*m3);

l1 = cvmGet(LINES,0,3);
l2 = cvmGet(LINES,1,3);
l3 = cvmGet(LINES,2,3);
m1 = cvmGet(LINES,0,0);
m2 = cvmGet(LINES,1,0);
```

```
m3 = cvmGet(LINES,2,0);
cvmSet(A, 3, 0, l1*m1);
cvmSet(A, 3, 1, (l1*m2+l2*m1)/2);
cvmSet(A, 3, 2, l2*m2);
cvmSet(A, 3, 3, (l1*m3+l3*m1)/2);
cvmSet(A, 3, 4, (l2*m3+l3*m2)/2);
cvmSet(A, 3, 5, l3*m3);

l1 = cvmGet(LINES,0,4);
l2 = cvmGet(LINES,1,4);
l3 = cvmGet(LINES,2,4);
m1 = cvmGet(LINES,0,5);
m2 = cvmGet(LINES,1,5);
m3 = cvmGet(LINES,2,5);
cvmSet(A, 4, 0, l1*m1);
cvmSet(A, 4, 1, (l1*m2+l2*m1)/2);
cvmSet(A, 4, 2, l2*m2);
cvmSet(A, 4, 3, (l1*m3+l3*m1)/2);
cvmSet(A, 4, 4, (l2*m3+l3*m2)/2);
cvmSet(A, 4, 5, l3*m3);

//-------------------------------------------------------------------------
//  computes the Cstar_inf matrix
//-------------------------------------------------------------------------
CvMat* V = cvCreateMat(6, 6, CV_64F);
CvMat* W = cvCreateMat(5, 6, CV_64F);
cvSVD(A, W, NULL, V, 0);
CvMat* Cstar_inf = cvCreateMat(3, 3, CV_64F);

cvmSet(Cstar_inf, 0, 0, cvmGet(V,0,5));         // a
cvmSet(Cstar_inf, 1, 1, cvmGet(V,2,5));         // c
cvmSet(Cstar_inf, 2, 2, cvmGet(V,5,5));         // f
cvmSet(Cstar_inf, 0, 1, 0.5*cvmGet(V,1,5));     // b/2
cvmSet(Cstar_inf, 1, 0, 0.5*cvmGet(V,1,5));     // b/2
cvmSet(Cstar_inf, 0, 2, 0.5*cvmGet(V,3,5));     // d/2
cvmSet(Cstar_inf, 2, 0, 0.5*cvmGet(V,3,5));     // d/2
cvmSet(Cstar_inf, 1, 2, 0.5*cvmGet(V,4,5));     // e/2
cvmSet(Cstar_inf, 2, 1, 0.5*cvmGet(V,4,5));     // e/2

if(cvmGet(Cstar_inf,2,2) < 0) {
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            cvmSet(Cstar_inf, i, j, -1.0*cvmGet(Cstar_inf, i, j));
        }
    }
}

//-------------------------------------------------------------------------
//  computes the homography
//-------------------------------------------------------------------------
CvMat* S = cvCreateMat(2, 2, CV_64F);
cvmSet(S,0,0,cvmGet(Cstar_inf,0,0));
cvmSet(S,0,1,cvmGet(Cstar_inf,0,1));
```

```
cvmSet(S,1,0,cvmGet(Cstar_inf,1,0));
cvmSet(S,1,1,cvmGet(Cstar_inf,1,1));

CvMat* U = cvCreateMat(2, 2, CV_64F);
CvMat* D2 = cvCreateMat(2, 2, CV_64F);
CvMat* D = cvCreateMat(2, 2, CV_64F);
CvMat* UD = cvCreateMat(2, 2, CV_64F);
CvMat* K = cvCreateMat(2, 2, CV_64F);

cvSVD(S, D2, U, NULL, 0);
cvPow(D2, D, 0.5);
cvMatMul(U, D, UD);
cvGEMM(UD, U, 1.0, NULL, 0, K, CV_GEMM_B_T);

CvMat* sol_vector = cvCreateMat(2, 1, CV_64F);
CvMat* v = cvCreateMat(2, 1, CV_64F);
CvMat* H = cvCreateMat(3, 3, CV_64F);
CvMat* Hinv = cvCreateMat(3, 3, CV_64F);

cvmSet(sol_vector, 0, 0, cvmGet(Cstar_inf, 0, 2));
cvmSet(sol_vector, 1, 0, cvmGet(Cstar_inf, 1, 2));
cvSolve(K, sol_vector, v, CV_LU);

cvZero(H);
for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; j++) {
        cvmSet(H, i, j, cvmGet(K, i, j));
    }
    cvmSet(H, 2, i, cvmGet(v, i, 0));
}
cvmSet(H, 2, 2, 1.0);
cvInvert(H, Hinv, CV_LU);

cvNormalize(H, H, 1, 0, CV_L2, NULL);
cvNormalize(Hinv, Hinv, 1, 0, CV_L2, NULL);

//---------------------------------------------------------------------
//  computes the bounds in the undistorted coordinate system
//---------------------------------------------------------------------
CvMat* bounds = cvCreateMat(1, 4, CV_64F);
compute_bounds(Hinv, image->height, image->width, bounds);
double xMin = cvmGet(bounds,0,0);
double xMax = cvmGet(bounds,0,1);
double yMin = cvmGet(bounds,0,2);
double yMax = cvmGet(bounds,0,3);

double scale_factor = ((double)(image->width))/(xMax-xMin);
int new_height = (int)((yMax-yMin)*(scale_factor));

//---------------------------------------------------------------------
//  removes the distortion from the image
//---------------------------------------------------------------------
IplImage *newImage = cvCreateImage(cvSize(image->width, new_height),
```

```
            IPL_DEPTH_8U, 3);
    cvZero(newImage);
    CvMat* Pw = cvCreateMat(3, 1, CV_64F);
    CvMat* Pi = cvCreateMat(3, 1, CV_64F);
    CvScalar pixel;

    double step = 1.0/scale_factor;
    int xi, yi;

    cvmSet(Pw, 2, 0, 1.0);
    for (i = 0; i < newImage->height; i++) {
        cvmSet(Pw, 1, 0, ((double)(i))*step+yMin);
        for (j = 0; j < newImage->width; j++) {
            cvmSet(Pw, 0, 0, ((double)(j))*step+xMin);
            cvMatMul(H, Pw, Pi);
            xi = ((int)(cvmGet(Pi,0,0)/cvmGet(Pi,2,0)));
            yi = ((int)(cvmGet(Pi,1,0)/cvmGet(Pi,2,0)));
            if (xi < 0 || xi >= image->width || yi < 0 ||
                    yi >= image->height)
            {
                continue;
            }
            pixel = cvGet2D(image,yi,xi);
            cvSet2D(newImage,i,j,pixel);
        }
    }

    cvSaveImage(outFile, newImage, 0);
    cvNamedWindow("corrected image", CV_WINDOW_AUTOSIZE);
    cvShowImage("corrected image", newImage);
    cvWaitKey(0);
    cvDestroyWindow("corrected image");

    //-------------------------------------------------------------------------
    //  release the images
    //-------------------------------------------------------------------------
    cvReleaseImage(&newImage);
    cvReleaseImage(&image);


    return 0;
}
```