Advanced Robot Navigation

1 Problem Description

In this Lab, our task is to inverse the distorted images to undistorted images by calculating the projective transform that is involved in the distorted images.

Projective distortion can be considered as a projective transform of the image. Let x $'$as distorted position and x as distortion free pixel position in homogenous representation. Then, the relation between x' and x will be as follows

$$x' = Hx \tag{1}$$

, where H is the projective transform matrix. Since matrix H has 8 degree of freedom, if we have 8 constraint equations, we can construct the matrix H as seen in the class.

$$\begin{aligned} h_{11}x + h_{12}y + h_{13} &= h_{31}xx' + h_{32}yx' + h_{33}x' && (2)\\ h_{21}x + h_{22}y + h_{23} &= h_{31}xy' + h_{32}yy' + h_{33}y'. && (3) \end{aligned}$$

After we construct H, we can do inverse the geometric distortion from the captured image.

$$x = H^{-1}x'. \tag{4}$$

Here, we can get the H^{-1} directly from eq.(4) since the projective transform matrix is in $PL(3)$. The matrix H^{-1} has also 8 degree of freedom and just like eq.(2) we can calculate the inverse projective matrix. To obtain the matrix $\hat{H} = H^{-1}$, we need 4 pixel positions initially to solve the following equation.

$$\begin{bmatrix} x'_1 & y'_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -y'_1y_1 \\ 0 & 0 & 0 & x'_1 & y'_1 & 1 & -x'_1y_1 & -y'_1y_1 \\ x'_2 & y'_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -y'_2y_2 \\ 0 & 0 & 0 & x'_2 & y'_2 & 1 & -x'_2y_2 & -y'_2y_2 \\ x'_3 & y'_3 & 1 & 0 & 0 & 0 & -x'_3x_3 & -y'_3y_3 \\ 0 & 0 & 0 & x'_3 & y'_3 & 1 & -x'_3y_3 & -y'_3y_3 \\ x'_4 & y'_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -y'_4y_4 \\ 0 & 0 & 0 & x'_4 & y'_4 & 1 & -x'_4y_4 & -y'_4y_4 \end{bmatrix} \begin{bmatrix} \hat{h}_{11} \\ \hat{h}_{12} \\ \hat{h}_{13} \\ \hat{h}_{21} \\ \hat{h}_{22} \\ \hat{h}_{23} \\ \hat{h}_{31} \\ \hat{h}_{32} \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix} \tag{5}$$

, where (x_i', y_i') are the distorted(captured image) 4 positions and (x_i, y_i) are the corresponding desired 4 positons.

From eq.(5), we can get the inverse projective matrix, $\hat{H}$. Then by simply applying eq.(4) we finally reconstruct the projective distortion free images.

2 Results

The captured images in fig.(1) and (3) show the geometric distortion. And the inverse projective transform as described is applied the images and we get the result images in fig.(2) and (4).

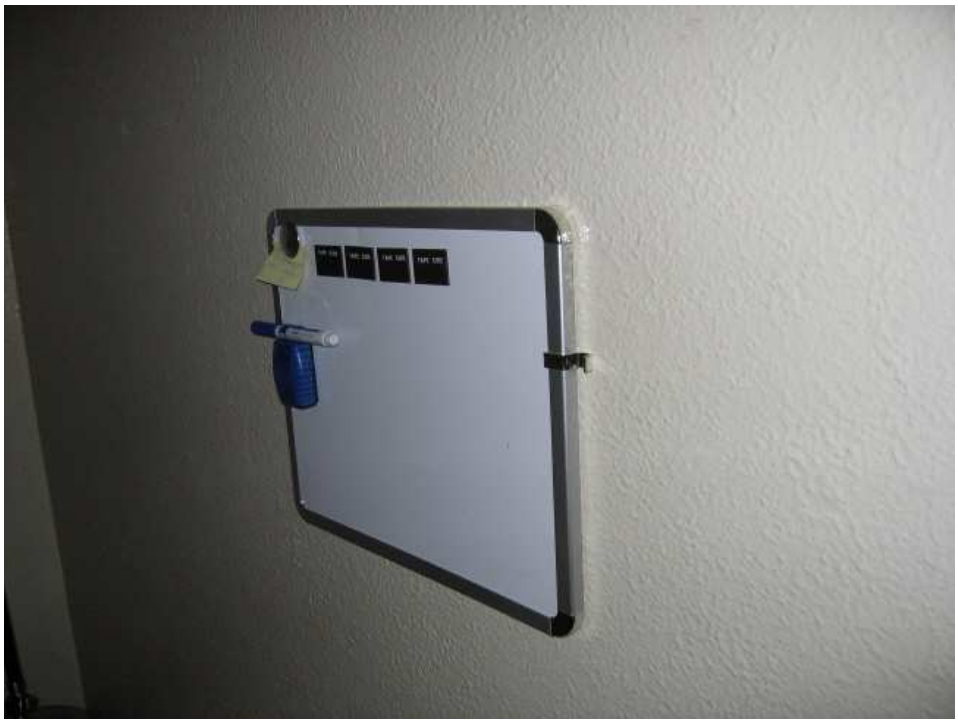

Figure 1: Geometrically distorted image.

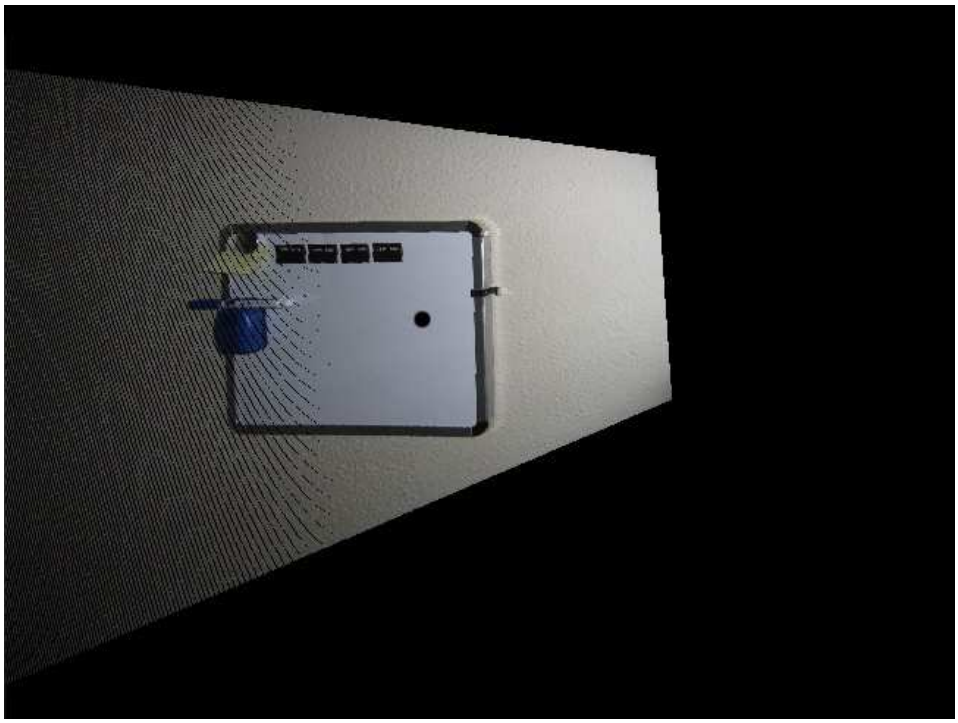

Figure 2: Reconstructed image.

Figure 3: Geometrically distorted image.

3 Code

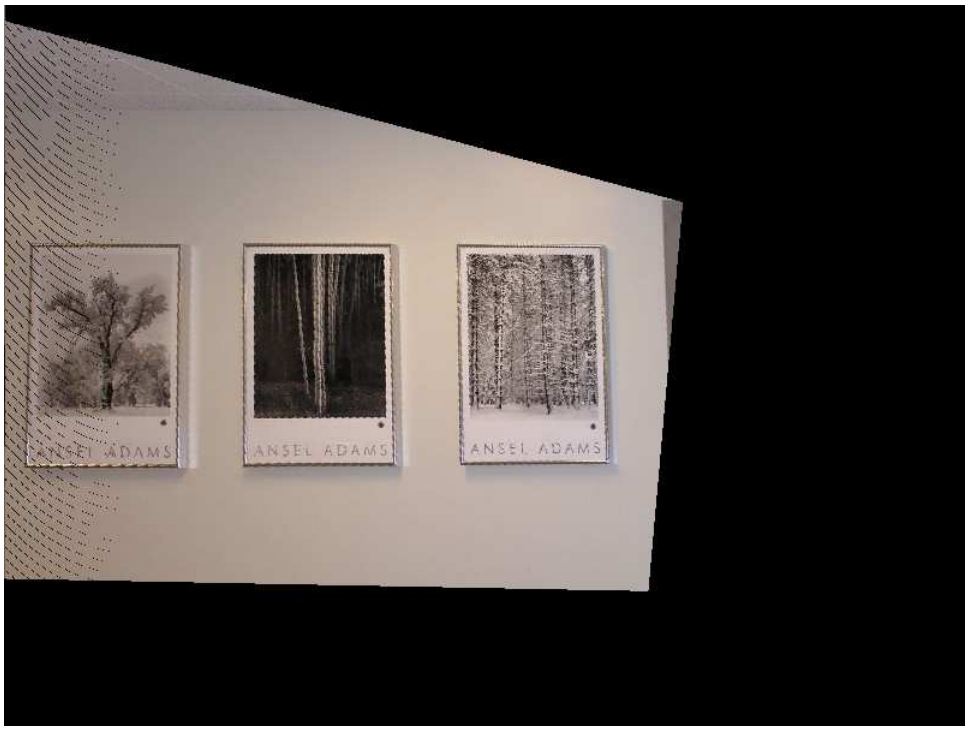

Figure 4: Reconstructed image.

```
//
// file : Lab1.cpp
// ---------------------------------------------------------------------------
// This main program converts a geometric distorted image to a front view image.
// First, it calculates the projective matrix from the 4 distorted and desired
// image positions.
// Then, using the homogeneuos representation, get undistorted pixel positions.
// ---------------------------------------------------------------------------
// onenCV library is utilized for image read and write and matrix calculations.
//
//

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

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

int main()
```

```
{
    // declaration
    IplImage *inImage = 0;
    IplImage *outImage = 0;
    int  inHeight, inWidth, inStep, inChannels;
    int  outHeight, outWidth, outStep, outChannels;
    uchar  *inData;
    uchar  *outData;
    int  i, j, k;
    int  inX1, inX2, inX3, inX4, inY1, inY2, inY3, inY4;
    float  outX1, outX2, outX3, outX4, outY1, outY2, outY3, outY4;
    float vLength, hLength; // target image's rectangular sizes
    char inImageName[80], outImageName[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);
     }

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

    // create the output image
    outHeight = inHeight;
    outWidth   = inWidth;
    outChannels = inChannels;
    outImage = cvCreateImage(cvSize(outWidth, outHeight), IPL_DEPTH_8U, outChannels);
    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;
            }
        }
    }
```

```
/*********************************************************/
/*              calculate the projective matrix          */
/*********************************************************/

// get the distorted image's 4 point positions
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);

// get the desired image's corresponding 4 point positions
printf("\n Type the 1st point in the desired image (x1, y1) : ");
scanf("%f, %f", &outX1, &outY1);
printf("\n Type vertical and horizontal line lengths (vLength, hLength) : ");
scanf("%f, %f", &vLength, &hLength);

outX2 = outX1 + hLength;
outY2 = outY1;
outX3 = outX1;
outY3 = outY1 + vLength;
outX4 = outX3 + hLength;
outY4 = outY3;
// calculate the matrix (x = Hx', where x : desired, x': ditorted)
float eqArrayM[64] = {inX1, inY1, 1, 0, 0, 0, -inX1*outX1, -inY1*outY1,
                      0, 0, 0, inX1, inY1, 1, -inX1*outY1, -inY1*outY1,
                      inX2, inY2, 1, 0, 0, 0, -inX2*outX2, -inY2*outX2,
                      0, 0, 0, inX2, inY2, 1, -inX2*outY2, -inY2*outY2,
                      inX3, inY3, 1, 0, 0, 0, -inX3*outX3, -inY3*outX3,
                      0, 0, 0, inX3, inY3, 1, -inX3*outY3, -inY3*outY3,
                      inX4, inY4, 1, 0, 0, 0, -inX4*outX4, -inY4*outX4,
                      0, 0, 0, inX4, inY4, 1, -inX4*outY4, -inY4*outY4};

float eqArrayV[8] = {outX1, outY1, outX2, outY2, outX3, outY3, outX4, outY4};
CvMat A  = cvMat(8, 8 , CV_32FC1, eqArrayM);
CvMat x  = cvMat(8, 1, CV_32FC1, eqArrayV);
CvMat b  = cvMat(8, 1, CV_32FC1, eqArrayV);
cvSolve(&A, &b, &x);

float h[8];
for(i = 0; i < 8; i++){
    h[i] = cvmGet(&x , i, 0);
```

```
    }
    float x1, x2, x3;
    int ii, jj;
    // apply the transform to get the target image
    for(i = 0; i < inHeight; i++){
        for(j = 0; j < inWidth; j++){
            for(k = 0; k < inChannels; k++){
                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)));
                outData[ii*outStep + jj*outChannels + k]
                = inData[i*inStep + j*inChannels + k];
            }
        }
    }
    // create a window
    cvNamedWindow("input image", CV_WINDOW_AUTOSIZE);
    cvMoveWindow("input imagen", 200, 200);
    cvNamedWindow("output image", CV_WINDOW_AUTOSIZE);
    cvMoveWindow("output imagen", 200, 200);

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

    // wait for a key
    cvWaitKey(0);

    // write output image
    sprintf(outImageName, "result.jpg");
    if(!cvSaveImage(outImageName, outImage)){
        printf("Could not save: %s\n",outImageName);
    }
    // release the images
    cvReleaseImage(&inImage);
    cvReleaseImage(&outImage);

    return 0;
}
```