Advanced Robot Navigation

Homography

1. Finding the Homography.

The transformation from the real world to the image plane can be described by the following equation:

$$x^{image} = Hx^{world}.$$

where x^{world} is the homogeneous representation of a point in the real world, x^{image} is the homogeneous representation of the point in the image plane, and H is the transformation matrix, which we call **Homography**.

As a homogeneous representation, H has 8 degree of freedom. Without loss of generality, we can write it as follows:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$

For one correspondence, we have

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

let

$$x' = \frac{x_1'}{x_3'}, y' = \frac{x_2'}{x_3'}, \text{and} x = \frac{x_1}{x_3}, y = \frac{x_2}{x_3}.$$

we have 2 equations:

$$x' = \frac{x_1'}{x_3'} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}$$

and

$$y' = \frac{x_2'}{x_3'} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}$$

Therefore, we can find the Homography using four correspondences, which give us the following matrix:

$$\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2'x_2 & -x_2'y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y_2'x_2 & -y_2'y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3'x_3 & -x_3'y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y_3'x_3 & -y_3'y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4'x_4 & -x_4'y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y_4'x_4 & -y_4'y_4 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}$$

Then we can solve this equation in OpenCV to get H.

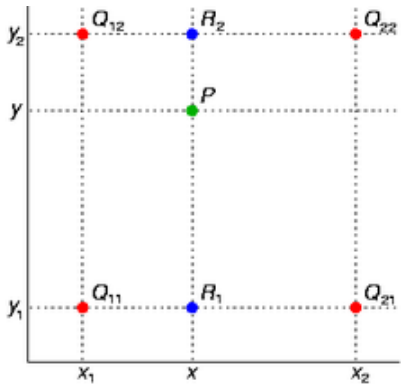

Figure 1: Bilinear Interpolation

2. Transform the image to the World Plane.

To get the real world image, basically we use the basic equation

$$x^{image} = Hx^{world}.$$

to map the real world points onto the image plane, find the corresponding points and get the value for RGB information.

First, we get the boundary points' corresponding position in the real world, and set the width of result image the same as the original image, so the height is set correspondingly.

Since every grid point in the world plane does not necessarily maps to a particular pixel of the image plane, so we use bilinear interpolation to smooth the results.

Bilinear interpolation is an extension of linear interpolation for interpolating functions of two variables on a regular grid. The key idea is to perform linear interpolation first in one direction,and then again in the other direction

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}), f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

and then

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location.

$$f(x, y) \approx \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{11}) + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{21}) + \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{12}) + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{22})$$

3. Images

3.1 Four Test Images

3.2 Three Images taken by me

Figure 2: Original board02 Image

Figure 3: Corrected board02 Image

Figure 4: Original door01 Image

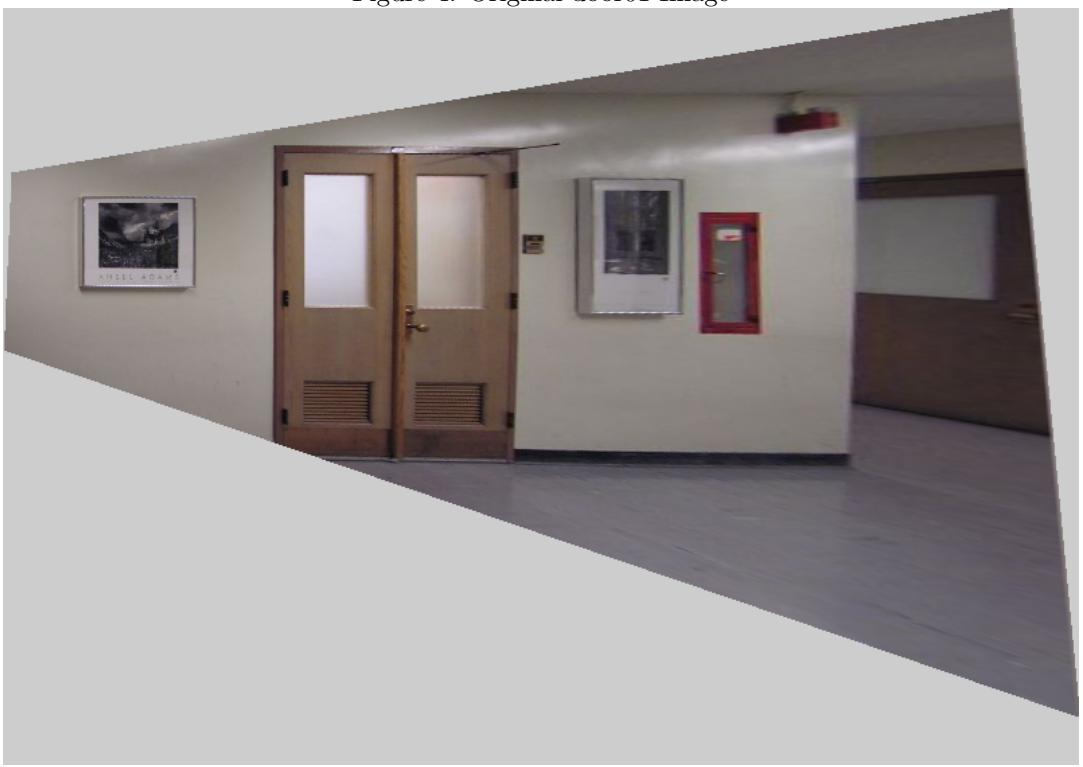

Figure 5: Corrected door01 Image

Figure 6: Original door02 Image

Figure 7: Corrected door02 Image

Figure 8: Original adams01 Image

Figure 9: Corrected adams01 Image

Figure 10: Original floor Image–taken by me

Figure 11: Corrected floor Image

Figure 12: Original dresser Image(taken by me)

Figure 13: Corrected dresser Image

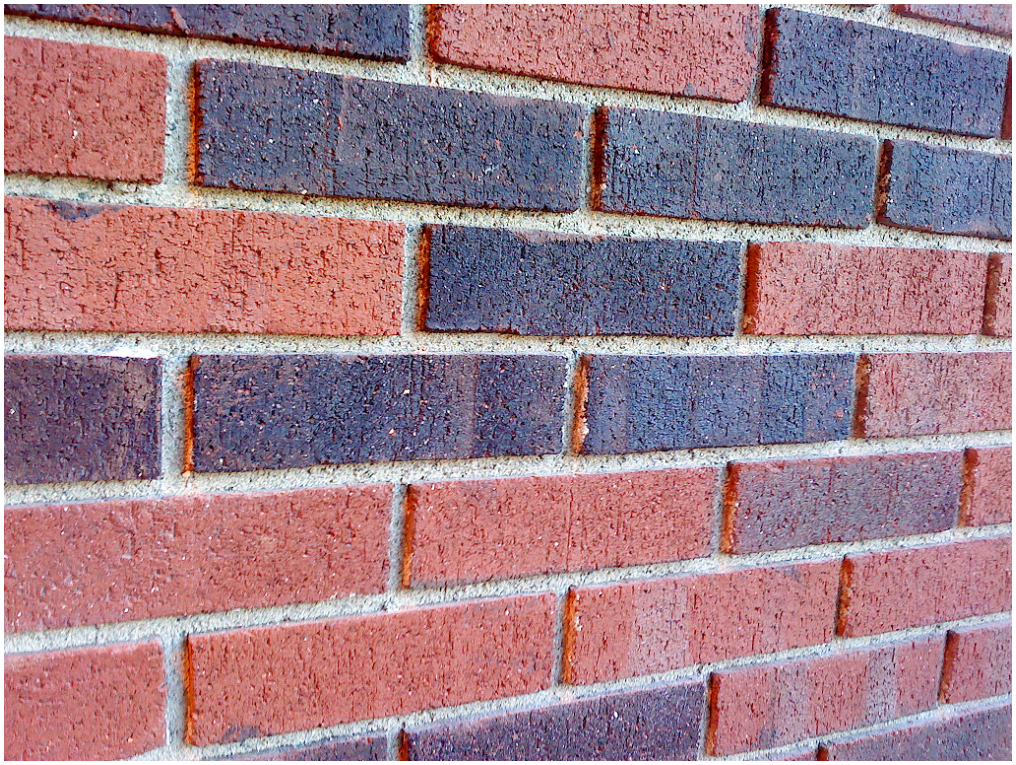

Figure 14: Original bricks Image(taken by me)

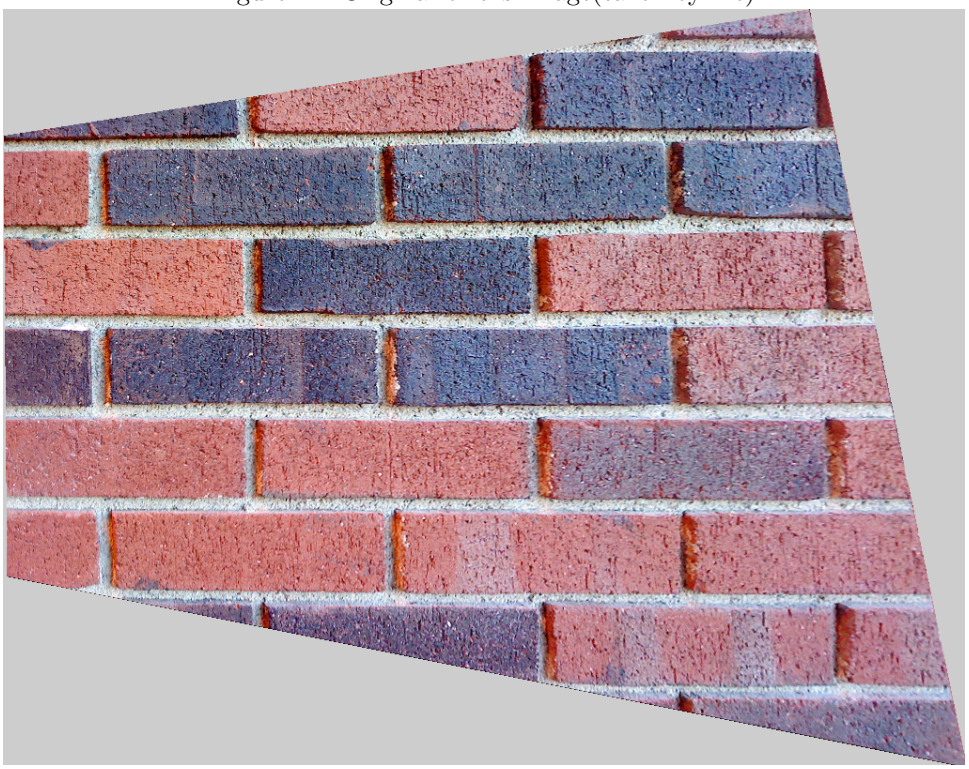

Figure 15: Corrected bricks Image

4. Source Code

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

int main(int argc, char* argv[])
{
	// Initialization

	char* name;
	// Check input arguments
	if (argc >= 2)
	{
		name = argv[1];
	}
	else
	{
		printf("Usage: filename \n");
		return 1;
	}


	// Define variables
	IplImage* original_image;
	IplImage* result_image;
	CvMat* HG;
	CvMat* HGI;
	int i, j, k;


	// Read the original image into memory
	if ( (original_image = cvLoadImage(name,1)) == 0)
	{
		printf("Error in loading the image! \n");
		return 2;
	}
```

```
// Input the 4 correspondance
for (i=0; i<4; i++)
{
	printf("Please input the world coordinate %d x,y \n", i+1);
	scanf("%lf,%lf", &input_world[2*i], &input_world[2*i+1]);
	printf("The input is: %lf,%lf \n", input_world[2*i], input_world[2*i+1]);

	printf("Please input the image coordinate %d x,y \n", i+1);
	scanf("%lf,%lf", &input_image[2*i], &input_image[2*i+1]);
	printf("The input is: %lf,%lf \n", input_image[2*i], input_image[2*i+1]);
}

// Calculate the Homography HG and its inverse HGI
double matrix[64] = {input_world[0], input_world[1], 1, 0, 0, 0,
	-input_image[0]*input_world[0], -input_image[0]*input_world[1],
	0, 0, 0, input_world[0], input_world[1], 1,
	-input_image[1]*input_world[0], -input_image[1]*input_world[1],
	input_world[2], input_world[3], 1, 0, 0, 0,
	-input_image[2]*input_world[2], -input_image[2]*input_world[3],
	0, 0, 0, input_world[2], input_world[3], 1,
	-input_image[3]*input_world[2], -input_image[3]*input_world[3],
	input_world[4], input_world[5], 1, 0, 0, 0,
	-input_image[4]*input_world[4], -input_image[4]*input_world[5],
	0, 0, 0, input_world[4], input_world[5], 1,
	-input_image[5]*input_world[4], -input_image[5]*input_world[5],
	input_world[6], input_world[7], 1, 0, 0, 0,
	-input_image[6]*input_world[6], -input_image[6]*input_world[7],
	0, 0, 0, input_world[6], input_world[7], 1,
	-input_image[7]*input_world[6], -input_image[7]*input_world[7]};
CvMat* M = cvCreateMat(8,8,CV_64FC1);
cvInitMatHeader(M,8,8,CV_64FC1,matrix,CV_AUTOSTEP);

double H[9];
CvMat* homo = cvCreateMat(8,1,CV_64FC1);
CvMat* point_i = cvCreateMat(8,1,CV_64FC1);
cvInitMatHeader(point_i,8,1,CV_64FC1,input_image,CV_AUTOSTEP);

// Solve the linear equations
cvSolve(M,point_i,homo, CV_LU);
for(i=0;i<8;i++)
{
	H[i] = cvmGet(homo,i,0);
}
H[8] = 1;
```

```
HG = cvCreateMat(3,3,CV_64FC1);
cvInitMatHeader(HG,3,3,CV_64FC1,H,CV_AUTOSTEP);

// Get the inverse of HG
HGI = cvCreateMat(3,3,CV_64FC1);
cvInvert(HG,HGI,0);

// print the HG
printf("HG= \n");
printf("%11.5lg %11.5lg %11.5lg \n", cvmGet(HG, 0, 0), cvmGet(HG, 0, 1), cvmGet(HG, 0, 2));
printf("%11.5lg %11.5lg %11.5lg \n", cvmGet(HG, 1, 0), cvmGet(HG, 1, 1), cvmGet(HG, 1, 2));
printf("%11.5lg %11.5lg %11.5lg \n\n", cvmGet(HG, 2, 0), cvmGet(HG, 2, 1), cvmGet(HG, 2, 2));
// print the HGI
printf("HGI= \n");
printf("%11.5lg %11.5lg %11.5lg \n", cvmGet(HGI, 0, 0), cvmGet(HGI, 0, 1), cvmGet(HGI, 0, 2));
printf("%11.5lg %11.5lg %11.5lg \n", cvmGet(HGI, 1, 0), cvmGet(HGI, 1, 1), cvmGet(HGI, 1, 2));
printf("%11.5lg %11.5lg %11.5lg \n\n", cvmGet(HGI, 2, 0), cvmGet(HGI, 2, 1), cvmGet(HGI, 2, 2));

// Transform the image to the world plane
// Determine the real-world points corresponds to boundaries of the image plane
CvMat* bound_w = cvCreateMat(3,4,CV_64FC1);
double bound_image[] = {0, original_image->width-1, 0, original_image->width-1,
        0, 0, original_image->height-1, original_image->height-1,
        1, 1, 1, 1};
CvMat* bound_i = cvCreateMat(3,4,CV_64FC1);
cvInitMatHeader(bound_i,3,4,CV_64FC1,bound_image, CV_AUTOSTEP);
cvMatMul(HGI, bound_i, bound_w);

double xx;
double yy;
double xmin = 0, ymin = 0, xmax = 0, ymax = 0;
for(i=0; i<4; i++)
{
        xx = cvmGet(bound_w,0,i)/cvmGet(bound_w,2,i);
        yy = cvmGet(bound_w,1,i)/cvmGet(bound_w,2,i);
        if(xx<xmin)
                xmin = xx;
        if(xx>xmax)
                xmax = xx;
        if(yy<ymin)
                ymin = yy;
        if(yy>ymax)
                ymax = yy;
}
```

```
// Allocate memory for the results
double scale = (original_image->width)/(xmax-xmin);
int new_height = (int)((ymax-ymin)*scale);

result_image = cvCreateImage(cvSize(original_image->width, new_height),IPL_DEPTH_8U, 3);
printf("Result image size is: %d, %d\n", result_image->width, result_image->height);

// Fill the corrected image
// (Use bilinear interpolation to smooth the results)
CvMat* pi = cvCreateMat(3,1,CV_64FC1);
CvMat* pw = cvCreateMat(3,1,CV_64FC1);
cvmSet(pw,2,0,1);

for(i=0; i<result_image->width; i++)
{
        cvmSet(pw, 0, 0,(double)i/scale+xmin);
        for(j=0; j<result_image->height; j++)
        {
                cvmSet(pw,1,0,(double)j/scale+ymin);
                cvMatMul(HG, pw, pi);
                xx = cvmGet(pi,0,0)/cvmGet(pi,2,0);
                yy = cvmGet(pi,1,0)/cvmGet(pi,2,0);

                if (xx<0 || yy<0 || xx>=original_image->width || yy>= original_image->height)
                        continue;

                // Do linear interpolation for all the 3 channels when constructing the results
                double sum;
                for(k=0; k<3; k++)
                {
                        sum = 0;
                        sum += (1.0-(xx-(int)xx))*(1.0-(yy-(int)yy))*((uchar*)(original_image->imageData +
original_image->widthStep*(int)yy))[((int)xx)*3+k];
                        sum += (1.0-(xx-(int)xx))*(yy-(int)yy)*((uchar*)(original_image->imageData +
original_image->widthStep*(int)(yy+1)))[((int)xx)*3+k];
                        sum += (xx-(int)xx)*(1.0 - (yy-(int)yy))*((uchar*)(original_image->imageData +
original_image->widthStep*(int)yy))[((int)(xx+1))*3+k];
                        sum += (xx-(int)xx)*(yy-(int)yy)*((uchar*)(original_image->imageData +
original_image->widthStep*(int)(yy+1)))[((int)(xx+1))*3+k];
                        ((uchar*)(result_image->imageData+result_image->widthStep*j))[i*3+k] = sum;
                }
        }
}
```

```
    // Display/Save results
    char* newname = "corrected_image.png";
    cvSaveImage(newname,result_image);
    cvNamedWindow("Original", CV_WINDOW_AUTOSIZE);
    cvShowImage("Original", original_image);
    cvNamedWindow("Results", CV_WINDOW_AUTOSIZE);
    cvShowImage("Results", result_image);
    cvWaitKey(0);

    // Release Memory
    cvReleaseImage(&original_image);
    cvReleaseImage(&result_image);
    cvDestroyWindow("Original");
    cvDestroyWindow("Results");
    cvReleaseMat(&HG);
    cvReleaseMat(&HGI);
    //cvReleaseMat(&point_w);
    cvReleaseMat(&M);
    cvReleaseMat(&homo);
    cvReleaseMat(&point_i);
    cvReleaseMat(&bound_w);
    cvReleaseMat(&bound_i);

    return 0;
}
```