ECE 661 HW 1

Chad Aeschliman

2008-09-09

1 Problem

The problem is to determine the homography which maps a point or line from a plane in the world to the image plane

$$\vec{x}^i = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \vec{x}^w \tag{1}$$

where h_{33} has been set to 1 without loss of generality and $\vec{x}^i$ and $\vec{x}^w$ are in $\mathcal{P}^3$. Four corresponding sets of coordinates in $\mathcal{R}^2$, (x_k^i, y_k^i) and (x_k^w, y_k^w), $k = 0, ..., 3$, are provided.

2 Solution

Plugging one set of corresponding equations into 1 gives the relationship

$$\begin{bmatrix} x_k^i \\ y_k^i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_k^w a_k^w \\ y_k^w a_k^w \\ a_k^w \end{bmatrix}$$

By carrying out the multiplication, dividing out the scale factor a_k^w, and simplifying we obtain two relationships

$$x_k^i = x_k^w h_{11} + y_k^w h_{12} + h_{13} - x_k^i x_k^w h_{31} - x_k^i y_k^w h_{31}$$
$$y_k^i = x_k^w h_{21} + y_k^w h_{22} + h_{23} - y_k^i x_k^w h_{31} - y_k^i y_k^w h_{31}$$

Combining the equations resulting from all four correspondences into a matrix equation gives the form

$$\begin{bmatrix} x_0^i \\ y_0^i \\ x_1^i \\ y_1^i \\ x_2^i \\ y_2^i \\ x_3^i \\ y_3^i \end{bmatrix} = \begin{bmatrix} x_0^w & y_0^w & 1 & 0 & 0 & 0 & -x_0^i x_0^w & -x_0^i y_0^w \\ 0 & 0 & 0 & x_0^w & y_0^w & 1 & -y_0^i x_0^w & -y_0^i y_0^w \\ x_1^w & y_1^w & 1 & 0 & 0 & 0 & -x_1^i x_1^w & -x_1^i y_1^w \\ 0 & 0 & 0 & x_1^w & y_1^w & 1 & -y_1^i x_1^w & -y_1^i y_1^w \\ x_2^w & y_2^w & 1 & 0 & 0 & 0 & -x_2^i x_2^w & -x_2^i y_2^w \\ 0 & 0 & 0 & x_2^w & y_2^w & 1 & -y_2^i x_2^w & -y_2^i y_2^w \\ x_3^w & y_3^w & 1 & 0 & 0 & 0 & -x_3^i x_3^w & -x_3^i y_3^w \\ 0 & 0 & 0 & x_3^w & y_3^w & 1 & -y_3^i x_3^w & -y_3^i y_3^w \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}$$

This matrix equation can then be solved in openCV to give H.

To construct the corrected image, the goal is to set up a grid of evenly spaced points in the world plane with each point representing one pixel in the corrected image and then mapping this grid of points to the image using H. The extents of the grid are determined by mapping the corners of the image plane to the world plane using H^{-1}. The smallest and largest x^w's and y^w's form the extents of the grid. The spacing of the grid points is chosen to set the resolution of the corrected image and the convention was chosen to preserve the width of the original image with the height adjusting as necessary.

In general a grid point does not map to a particular pixel in the original image so linear interpolation was used.

3 Results

8 pairs of original and corrected images are included at the end after the code.

4 Code

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

// utility function which prints out a matrix
void printMatrix(CvMat* M, int rows, int cols, const char* name)
{
    int indent_size;
    int i,j;

    // print the matrix name
    indent_size = printf("%s = ", name);

    // print out the matrix
    for (i=0;i<rows;i++)
    {
        // start of a row
        printf("[");
        for (j=0;j<cols;j++)
        {
            if (j>0)
                printf(",");
            printf("%11.5lg",cvmGet(M,i,j));
        }
        printf("]\n");

        // indent the next line
        for (j=0;j<indent_size;j++)
        {
            printf(" ");
        }
    }
```

```
    printf("\n");
}

// main function, reads in an image file and corresponding
// coordinates and computes a perspective corrected image
int main( int argc, char** argv )
{
    char* filename;

    IplImage* image;
    IplImage* corrected_image;
    CvMat* H;
    CvMat* invH;

    int i,j,k;
    double xw[4];
    double yw[4];
    double xi[4];
    double yi[4];

    // attempt to read in the image file
    if (argc >= 2)
    {
        filename = argv[1];
    }
    else
    {    cvReleaseImage(&image);

        printf("Usage: hw1 filename");
        return 1;
    }
    if((image = cvLoadImage(filename,1)) == 0)
    {
        printf("Could not read the file!");
        return 2;
    }

    // read user input of the corresponding coordinates in the
    // world and image
    for (i=0; i<4; i++)
    {
        printf("Enter world coordinate %d as x,y: ",i);
        scanf("%lf,%lf",xw+i,yw+i);
        printf("Enter image coordinate %d as x,y: ",i);
        scanf("%lf,%lf",xi+i,yi+i);
    }
    printf("\n");

    // set up the problem as a matrix equation
    double sol_matrix[64];
    double sol_vector[8];
    for (i=0;i<4;i++)
    {
        sol_matrix[2*i*8+0] = xi[i];
        sol_matrix[2*i*8+1] = yi[i];
        sol_matrix[2*i*8+2] = 1;
        sol_matrix[2*i*8+3] = 0;
```

```
    sol_matrix[2*i*8+4] = 0;
    sol_matrix[2*i*8+5] = 0;
    sol_matrix[2*i*8+6] = -xw[i]*xi[i];
    sol_matrix[2*i*8+7] = -xw[i]*yi[i];

    sol_matrix[(2*i+1)*8+0] = 0;
    sol_matrix[(2*i+1)*8+1] = 0;
    sol_matrix[(2*i+1)*8+2] = 0;
    sol_matrix[(2*i+1)*8+3] = xi[i];
    sol_matrix[(2*i+1)*8+4] = yi[i];
    sol_matrix[(2*i+1)*8+5] = 1;
    sol_matrix[(2*i+1)*8+6] = -yw[i]*xi[i];
    sol_matrix[(2*i+1)*8+7] = -yw[i]*yi[i];

    sol_vector[2*i] = xw[i];
    sol_vector[2*i+1] = yw[i];
}

// solve the problem and copy the solution into invH
CvMat* temp = cvCreateMat(8,1,CV_64FC1);
invH = cvCreateMat(3,3,CV_64FC1);
CvMat A;
CvMat B;
cvInitMatHeader(&A,8,8,CV_64FC1,sol_matrix,CV_AUTOSTEP);
cvInitMatHeader(&B,8,1,CV_64FC1,sol_vector,CV_AUTOSTEP);
cvSolve(&A, &B, temp, CV_LU);
for (i=0;i<8;i++)
{
    cvmSet(invH,i/3,i%3,cvmGet(temp,i,0));
}
cvmSet(invH,2,2,1.0);
cvReleaseMat(&temp);

// get H from H^-1
H = cvCreateMat(3,3,CV_64FC1);
cvInvert(invH,H,CV_LU);

// display H and H^-1
printMatrix(invH,3,3,"H^-1");
printMatrix(H,3,3,"H");


// we need to determine the real-world bounds that the
// image represents
double max_x = -1.0e100;
double max_y = -1.0e100;
double min_x = 1.0e100;
double min_y = 1.0e100;
CvMat* world_coord = cvCreateMat(3,4,CV_64FC1);
CvMat* image_coord = cvCreateMat(3,4,CV_64FC1);

// pick some representative image coordinates
cvmSet(image_coord,0,0,0);
cvmSet(image_coord,1,0,0);
cvmSet(image_coord,2,0,1);

cvmSet(image_coord,0,1,0);
```

```
cvmSet(image_coord,1,1,image->height-1);
cvmSet(image_coord,2,1,1);

cvmSet(image_coord,0,2,image->width-1);
cvmSet(image_coord,1,2,0);
cvmSet(image_coord,2,2,1);

cvmSet(image_coord,0,3,image->width-1);
cvmSet(image_coord,1,3,image->height-1);
cvmSet(image_coord,2,3,1);

// apply H^-1 to get the corresponding world coordinates
cvMatMul(invH,image_coord,world_coord);

// now check the bounds
for (i=0;i<4;i++)
{
   double w = cvmGet(world_coord,2,i);
   double real_x = cvmGet(world_coord,0,i)/w;
   double real_y = cvmGet(world_coord,1,i)/w;

   if (real_x < min_x)
      min_x = real_x;
   if (real_y < min_y)
      min_y = real_y;
   if (real_x > max_x)
      max_x = real_x;
   if (real_y > max_y)
      max_y = real_y;

}

// determine the correct size for the new image assuming that
// the new width should match the old width
double scale_factor = ((double)(image->width))/(max_x-min_x);
int new_height = (int)((max_y-min_y)*scale_factor);
corrected_image = cvCreateImage(cvSize(image->width,new_height),
                                IPL_DEPTH_8U, 3);
cvZero(corrected_image);

// compute the new image by applying H to a grid of points in
// the world coordinate system
double step_size = 1.0/scale_factor;
CvMat* input_coord = cvCreateMat(3,1,CV_64FC1);
CvMat* output_coord = cvCreateMat(3,1,CV_64FC1);
cvmSet(input_coord,2,0,1.0);
for (i=0; i<corrected_image->width; i++)
{
   cvmSet(input_coord,0,0,((double)i)*step_size+min_x);
   for (j=0; j<corrected_image->height; j++)
   {
      double xi,yi,fx,fy;
      cvmSet(input_coord,1,0,((double)j)*step_size+min_y);

      // compute the associated image coordinate
      cvMatMul(H,input_coord,output_coord);
      xi = cvmGet(output_coord,0,0)/cvmGet(output_coord,2,0);
```

```
            yi = cvmGet(output_coord,1,0)/cvmGet(output_coord,2,0);

            // if outside of the image then move on
            if (xi<0||yi<0||xi>=(image->width-1)||yi>=(image->height
                -1))
            {
               continue;
            }

            // compute the fractional component of the image coord.
            fx = xi - (int)xi;
            fy = yi - (int)yi;

            // compute the pixel value using linear interpolation
            for (k=0;k<3;k++)
            {
               double value = 0;
               value += (1.0-fx)*(1.0-fy)*((uchar*)(image->imageData +
                    image->widthStep*(int)yi))[((int)xi)*3+k];
               value += (1.0-fx)*fy*((uchar*)(image->imageData + image
                   ->widthStep*(int)(yi+1)))[((int)xi)*3+k];
               value += fx*(1.0-fy)*((uchar*)(image->imageData + image
                   ->widthStep*(int)yi))[((int)(xi+1))*3+k];
               value += fx*fy*((uchar*)(image->imageData + image->
                    widthStep*(int)(yi+1)))[((int)(xi+1))*3+k];
              ((uchar*)(corrected_image->imageData + corrected_image->
                   widthStep*j))[i*3+k] = value;
            }
         }
      }

      // save corrected image
      char new_filename[128];
      sprintf(new_filename,"%s.corrected.png",filename);
      cvSaveImage(new_filename,corrected_image);

      // cleanup
      cvReleaseImage(&image);
      cvReleaseImage(&corrected_image);
      cvReleaseMat(&H);
      cvReleaseMat(&invH);
      cvReleaseMat(&world_coord);
      cvReleaseMat(&image_coord);
      cvReleaseMat(&input_coord);
      cvReleaseMat(&output_coord);

      return 0;

}
```

Figure 1: adams01

Figure 2: adams01.corrected

Figure 3: adams03

Figure 4: adams03.corrected

Figure 5: board01

Figure 6: board01.corrected

Figure 7: door01

Figure 8: door01.corrected

Figure 9: bricks

Figure 10: bricks.corrected

Figure 11: clock

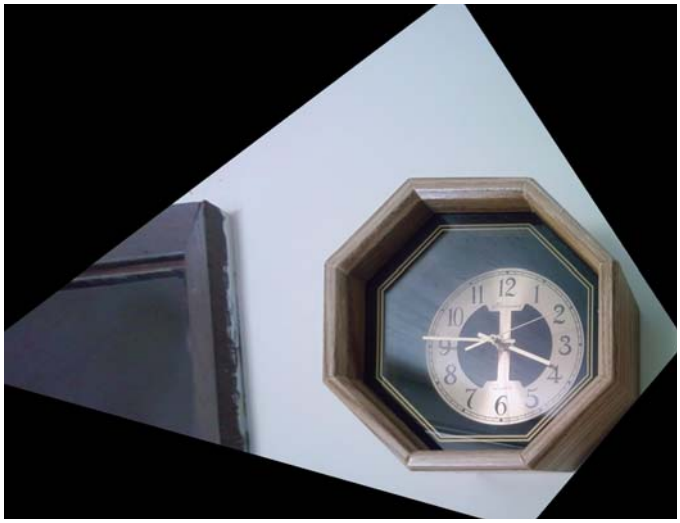

Figure 12: clock.corrected

Figure 13: mailboxes

Figure 14: mailboxes.corrected

Figure 15: piano

Figure 16: piano.corrected