Problem 1. (60 points)

The goal is for you to apply your knowledge of Homography estimation from a set of image features in order to perform a simple image warping task. In particular, you are expected to implement

a) The DLT algorithm for homography estimation

b) 2D Bilinear interpolation to render the output image

basketball-court.png	9/21/2017 10:11 A...	PNG File	689 KB
basketball-court.ppm	9/14/2017 8:09 PM	PPM File	524 KB
positions.mat	9/19/2017 3:00 PM	MATLAB Data	1 KB
problem1.m	9/21/2017 10:20 A...	MATLAB Code	2 KB

As the front picture, the code path of this problem is: *Assignment1**Problem 1**problem1.m* . The chosen points have been save in a *.mat file* : *Assignment1**Problem 1**points.mat* which contains 7 key points. With the corresponding, there are also 7 points in the transformed image.

22	192	1	1	1	1
248	50	1	940	1	1
401	72	1	940	500	1
281	281	1	1	500	1
168	101	1	470	1	1
258	120	1	470	250	1
363	142	1	470	500	1

Figure 1 the original key points positions(left) and the transformed key points positions(right)

Here is the relevant code:

```
PointStruct = load('positions.mat');
corrdinates = PointStruct.corrdinates;
    X_ = zeros(7,3);
    for i = 1 : 7
        X_(i,1:2) = corrdinates(8 - i).Position;
        X_(i,3) = 1;
    end
    X = [1,1,1;
        940,1,1;
        940,500,1;
        1,500,1;
        470,1,1;
        470,250,1;
        470,500,1];
```

Following the arithmetic, 2 sets of points are put into the formula and then we can get a 14(2x7) x 9 matrix From this matrix, we calculate the matrix H.

-0.0024	-0.0020	-0.1105
5.535...	-2.790...	-0.9939
-4.636...	2.213...	-0.0051

Figure 2 the matrix H

Here is the relevant code:

```
M = zeros(14,9);
    for i = 1 : 7
        M(2*i - 1 , 1 : 3) = 0;
        M(2*i - 1 , 4 : 6) = -X_(i,3) * X(i,:);
        M(2*i - 1 , 7 : 9) = X_(i,2) * X(i,:);
        M(2*i , 1 : 3) = X_(i,3) * X(i,:);
        M(2*i , 4 : 6) = 0;
        M(2*i , 7 : 9) = -X_(i,1) * X(i,:);
    end
    H = zeros(3,3);
    [U,S,V] = svd(M);
    M_ = V(:,9);
    for i = 1 : 3
        for j = 1 : 3
            H(i,j) = M_((i-1)*3+j,1);
        end
    end
```

The following line of code is just for testing to insure that the H we get is right:

```
%     % for testing
%             p = H*[940;500;1];
%             i_ = int16(p(1,1)/p(3,1))
%             j_ = int16(p(2,1)/p(3,1))
%     %end
```

When we bring the H into X` = H*X, we can the original point position correspondingly. Then we copy every original color matrix to the new one so that we get the transformed image.

Here is the relevant code:

```
    for i = 1 : 940
        for j = 1 : 500
            p = H*[i;j;1];
            i_ = p(1,1)/p(3,1);
            j_ = p(2,1)/p(3,1);
            if i_> 0 && j_ > 0
                i_int = fix(i_);
                j_int = fix(j_);
                dif_i = i_ - i_int;
                dif_j = j_ - j_int;
                pic(j,i,:) =(1-dif_i)*(1-dif_j)*Pic(j_int,i_int,:)...
                            + dif_i*dif_j*Pic(j_int+1,i_int+1,:)...
                            + dif_i*(1-dif_j)*Pic(j_int+1,i_int,:)...
                            + dif_j*(1-dif_i)*Pic(j_int,i_int+1,:);
```

```
            end
        end
    end
```

In every loop, we do the 2D Bilinear interpolation and abandon the redundant points which are outside the court.

Finally we get the transformed image:

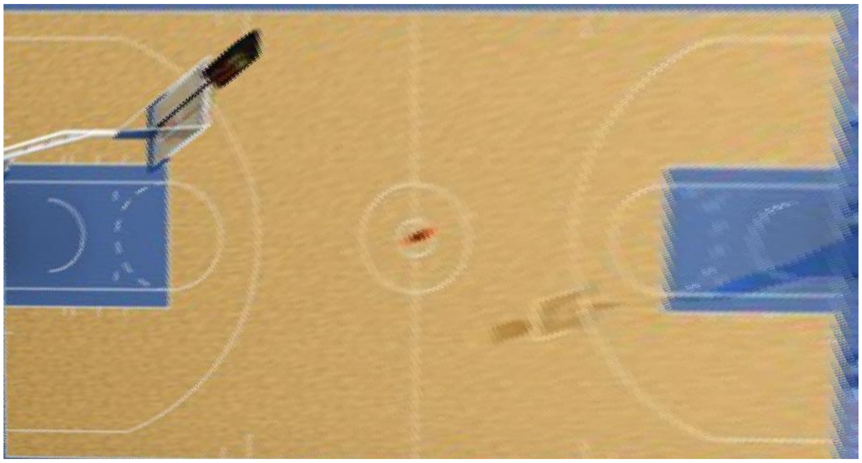

Figure 3 the result image

Problem 2. (40 points)

Dolly Zoom The goal is for you to apply your knowledge of the pinhole camera model by controlling both the internal and external parameters of a virtual camera in order to simulate the effect of a "dolly zoom". The dolly zoom is an optical effect used by cinematographers. The effect consists in adjusting the distance of the camera to a foreground object in the scene, while simultaneously controlling the camera's field of view (a function of the focal length), in order for the foreground object to retain a constant size in the image throughout the entire capture sequence.

From calculating the cone of vision of the camera, the f should be enlarged 1.6 times in every loop so that the field of view can be smaller, and the projection matrix is made up by f, so we can finish the task by changing it. Here is the code:

```
clc
clear all
% load variables:
BackgroundPointCloudRGB,ForegroundPointCloudRGB,K,crop_region,filter_size)
load data.mat

data3DC = {BackgroundPointCloudRGB,ForegroundPointCloudRGB};
K(1,1) = K(1,1)*1.6;
K(2,2) = K(2,2)*1.6;
f = (K(1,1)+K(2,2))/2;
R       = eye(3);
move    = [0 0 -0.25]';
% dis = f/400;
dis = 13;


for step=0:15
    tic
```

```
    fname        = sprintf('SampleOutput%03d.jpg',step);
    display(sprintf('\nGenerating %s',fname));
    K(1,1)       = (dis+move(3)*step)*K(1,1)/dis+move(3)*(step-1);
    K(2,2)       = (dis+move(3)*step)*K(2,2)/dis+move(3)*(step-1);
    t            = step*move;
    M            = K*[R t];
    im           = PointCloud2Image(M,data3DC,crop_region,filter_size);
    imwrite(im,fname);
    toc
end
```

The code's path is *Assignment1\Problem 2\Dolly_Data_Code\Dolly_Data_Code\problem2,* it is modified from the sample code. In every loop, we change the moving matrix and change the projection matrix, so the background seems go backer but the foreground doesn't move.