EECS 442 Homework #3

Yi Yang

10/23/2016

Problem 1

(a)

Suppose in the world coordinate, there exists a line l_0 passing through point A, then any point in this line can be represented as $B = A + rd$, $\forall B \in l_0$. Transforming B to homogeneous coordinate, we have:

$$B_w = \begin{bmatrix} a_1 + r\cos\alpha \\ a_2 + r\cos\beta \\ a_3 + r\cos\gamma \\ 1 \end{bmatrix} \cong \begin{bmatrix} \frac{a_1}{r} + \cos\alpha \\ \frac{a_2}{r} + \cos\beta \\ \frac{a_3}{r} + \cos\gamma \\ \frac{1}{r} \end{bmatrix}$$

We know that the vanishing point is obtained when $r \to \infty$, hence

$$\lim_{r\to\infty} B_w = \begin{bmatrix} \cos\alpha \\ \cos\beta \\ \cos\gamma \\ 0 \end{bmatrix}$$

Then the image coordinates of the vanishing point can be written as:

$$v = K[R\ T] \lim_{r\to\infty} B_w = KRd$$

where d is given by $d = \begin{bmatrix} \cos\alpha \\ \cos\beta \\ \cos\gamma \end{bmatrix}$, K is the camera calibration matrix and R is the rotation matrix between the camera and the world coordinates.

(b)

First we must prove that K and R is nonsingular, in lecture we have shown that K has the form as follows:

$$K = \begin{bmatrix} fl & fl\cot\theta & u_0 \\ 0 & \frac{fk}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then, the determinant of K is $|K| = \frac{f^2 k}{\sin\theta} \neq 0$, which means K is nonsingular and it has inverse matrix. Next, for rotation matrix R, we have:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Hence the determinant of R is $det(R) = det(R_x) \cdot det(R_y) \cdot det(R_z) = 1 \times 1 \times 1 = 1 \neq 0$, that means R is nonsingular and inverse matrix of R exists. Therefore, we get the final expression of d in terms of K, R and v.

$$d = R^{-1}K^{-1}v$$

(c)

From section (b), we can easily prove that R is a unitary matrix(orthogonal matrix), which has the property that $RR^T = R^TR = I$. Since $Rd = K^{-1}v$, we have:

$$\left(K^{-1}v_i\right)^T \left(K^{-1}v_j\right) = d_i^T R^T R d_j = d_i^T d_j$$

Using the conditions provided in the question, we have:

$$\left(K^{-1}v_i\right)^T \left(K^{-1}v_j\right) = 0$$

for $i \neq j$.

Problem 2

(a)

No, since the camera matrix will not be full rank when we pick only one checkboard for camera calibration. The vertex matrix is nor invertible and we can not calculate the inverse of it. We can calibrate the matrix only if we can use pseudoinverse to the compute the solution of a underdetermined linear system equations.

(b)(c)

We can first express n group of correspondences in affine form and put all the unknowns in the coefficient matrix P into a unknown vector x and re-regulate the system equations, then we can get a coefficient matrix with known parameters. The last step is to apply least square approximation to the new formulated system of equations and get the final results. The original system can be represented as:

$$x = \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ 1 & \cdots & 1 \end{bmatrix}, \quad P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} X_1 & \cdots & X_n \\ Y_1 & \cdots & Y_n \\ Z_1 & \cdots & Z_n \\ 1 & \cdots & 1 \end{bmatrix}$$

By expanding out the first 3D world to 2D point correspondences, we obtain:

$$x_1 = p_{11}X_1 + p_{12}Y_1 + p_{13}Z_1 + p_{14}$$

$$y_1 = p_{21}X_1 + p_{22}Y_1 + p_{23}Z_1 + p_{24}$$

Use the similar way, we can represent all the corresponding points in a system equations represented as:

$$Ax = b = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 \\ \cdots & & & & & & & \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \cdots \end{bmatrix}$$

Hence, we can solve the equation listed above by apply least square approximation to it and we can get:

$$x = (A^T A)^{-1} A^T b$$

We can also directly solve for P via the original system of equations by minimizing the norm $\|PX - x\|$, we get the following solutions:

$$P = xX^T(XX^T)^{-1}$$

Then, we can use the projective matrix we have solved for to compute the RMS error which is defined as:

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left((x_i - x_{i_{calculated}})^2 + (y_i - y_{i_{calculated}})^2\right)}$$

where N is the number of correspondences we use for approximation in this problem.
We calculate that the RMS error (in pixels) between the corner positions and the positions predicted by the camera parameters is:

$$RMS = 2.2931$$

The 3×4 calibrated camera matrix is:

$$P = \begin{bmatrix} 1.3480 & 0.0497 & -0.0970 & 160.1158 \\ -0.0442 & 1.3396 & 0.2547 & 60.8333 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

The Matlab codes for computation is attached below:

```
% EECS442 Problem 2, Homework 3
% Linear Fit for Perspective Camera Model Calibration
% Code overview: Load in data, extract corners, create correspondences,then
% solve for the missing parameters via least squares
% by Yi Yang
% Date: 10/25/2016

%% Load in images
img1 = imread('homework3/Q2/1.JPG');
img2 = imread('homework3/Q2/2.JPG');
figure, imagesc(img1); colormap(gray);
title('Click a point on this image'); axis image;
a = []

%% Take in corner input - make sure to click in order corresponding to A
%% below.

for i = 1:12
[x y] = ginput(1);
a = [a [x;y]]
img1(round(y),round(x)) = 255;
imagesc(img1); colormap(gray);
title('Click a point on this image'); axis image;
end

figure, imagesc(img2); colormap(gray);
title('Click a point on this image'); axis image;
for i = 1:12
[x y] = ginput(1);
a = [a [x;y]]
img1(round(y),round(x)) = 255;
imagesc(img2); colormap(gray);
title('Click a point on this image'); axis image;
end
```

```
save a
%% 3D coordinates
A  = [];
A = [ [ 0 0 0]' [35 0 0]' [70 0 0]' [105 0 0]' [140 0 0]' [175 0 0]' ...
[175 35 0]' [175 70 0]' [175 105 0]' [175 140 0]' [175 175 0]' [192.5 192.5 0]' ];
A = [ A [ 0 0 9.5]' [35 0 9.5]' [70 0 9.5]' [105 0 9.5]' [140 0 9.5]' [175 0 9.5]' ...
[175 35 9.5]' [175 70 9.5]' [175 105 9.5]' [175 140 9.5]' [175 175 9.5]' [192.5 192.5 ...
    9.5]' ];
A = [A;ones(1,size(A,2)) ]
aFix = [a; ones(1, size(a,2))]

%Solve for the matrix (least norm)
% P = aFix*A'*inv(A*A')

%Solve using least squares
P = findHMatrixCalibration(A', a');

%RMS Calculation
aCalculated = P * A
RMS = sqrt(mean(sum((aCalculated - aFix).^2,1)))


% RMS =
%
%     2.2931



%-------------------------------------------------------------
%returns the H matrix given point (3D) and pointp(image) (transformed coordinates)
%   point is N X 3,   pointp is N X 2
```

We use findHMatrixCalibration.m function to apply least square algorithm to calculate vectorized P. Codes are listed below:

```
function out = findHMatrixCalibration (point, pointp)

%shows  the general form of A matrix...
% x1 = point1(1);
% x2 = point2(1);
% x3 = point3(1);
% x4 = point4(1);
% y1 = point1(2);
% y2 = point2(2);
% y3 = point3(2);
% y4 = point4(2);
%
% x1p = point1p(1);
% x2p = point2p(1);
% x3p = point3p(1);
% x4p = point4p(1);
% y1p = point1p(2);
% y2p = point2p(2);
% y3p = point3p(2);
% y4p = point4p(2);
%
% A = [x1 y1 z1 1 0  0  0  0;
%      0  0  0 0 x1 y1 z1 1;
%      x2 y2 z2 1 0  0  0  0;
%      0  0  0 0 x2 y2 z2 1 ;
%      x3 y3 z3 1 0  0  0  0;
%      0  0  0 0 x3 y3 z3 1;
```

```
%       x4 y4 z4 1 0  0  0  0;
%       0  0  0 0 x4 y4 z4 1  ;
%       ];
%  ...
% b = [x1p y1p x2p y2p x3p y3p x4p y4p ... ]';
%
A = [];
b = [];
for index = 1:size(point,1)
    xCurrent = point(index,1);
    xpCurrent = pointp(index,1);
    yCurrent = point(index,2);
    ypCurrent = pointp(index,2);
    zCurrent =point(index, 3);


    %calculates A matrix and B vector
    A = [A;
         xCurrent yCurrent zCurrent 1 0 0 0 0;
         0 0 0 0 xCurrent yCurrent zCurrent 1];
     b = [b; xpCurrent; ypCurrent];
end

%least squares solution
 x = (A'*A)^-1 * A'*b;
%  x = A\B;  This is an alternative way to obtain the solution.

 a = x(1);
 b = x(2);
 c = x(3);
 d = x(4);
 e = x(5);
 f = x(6);
 g = x(7);
 h = x(8);

 H = [a b c d;
      e f g h;
      0 0 0 1];
 out = H;

end
```