# CS 231A Computer Vision, Spring 2016
# Problem Set 1

Due Date: April 18th 2016 11:59 pm

## 1 Projective Geometry Problems [10 points]

In this question, we will examine properties of projective transformations. We define a camera coordinate system, which is only rotated and translated from a world coordinate system.

**1.a [2 points]** Prove that parallel lines in the world reference system are still parallel in the camera reference system.

**1.b [2 points]** Consider a unit square $pqrs$ in the world reference system where $p, q, r$, and $s$ are points. Will the same square in the camera reference system always have unit area? Prove or provide a counterexample.

**1.c [3 points]** Now let's consider affine transformations, which are any transformations that preserve parallelism. Affine transformations include not only rotations and translations, but also scaling and shearing. Given some vector $p$, an affine transformation is defined as

$$A(p) = Mp + b$$

where $M$ is an invertible matrix. Prove that under any affine transformation, the ratio of parallel line segments is invariant, but the ratio of non-parallel line segments is not invariant.

**1.d [3 points]** You have explored whether these three properties hold for affine transformations. Do these properties hold under any projective transformation? Justify briefly (no proof needed).

**Solution:**

(a) Consider any two parallel lines $k$ and $l$. Take any two points $k_1, k_2$ on $k$ and two points $l_1, l_2$ on l. By definition of parallel lines, $(k_1 - k_2) \times (l_1 - l_2) = 0$. If translated by any vector $p$, we see that

$$(k_1 + p - k_2 - p) \times (l_1 + p - l_2 - p) = (k_1 - k_2) \times (l_1 - l_2) = 0$$

If rotated by any rotation matrix $R$, we see that

$$(Rk_1 - Rk_2) \times (Rl_1 - Rl_2) = R(k_1 - k_2) \times R(l_1 - l_2) = R((k_1 - k_2) \times (l_1 - l_2)) = 0$$

Thus, parallel lines are invariant to rotation and translation.

(b) We are given that the square $pqrs$ has square unit area. This means that

$$\|(q-p)\times(s-p)\| = 1$$

In the new coordinate system, we multiply all points by a transformation matrix T, giving $Tp, Tq, Tr$, and $Ts$. Thus, we have to find $\|(Tq-Tp)\times(Ts-Tp)\|$. Because transformation matrix T is isometric (and consequently a determinant of 1), then we see that

$$\|(Tq-Tp)\times(Ts-Tp)\| = (\det T)\|(q-p)\times(s-p)\| = 1\cdot\|(q-p)\times(s-p)\| = 1$$

Therefore, the same square in the camera reference system will always have unit area.

(c) Consider any two parallel lines $k$ and $l$. Take the segment between any two points $k_1, k_2$ on $k$ and the segment between any two points $l_1, l_2$ on l. By definition of parallel segments,

$$k_1 - k_2 = k(l_1 - l_2)$$

for some real number $k$. Thus,

$$\|k_1 - k_2\| = k\|l_1 - l_2\|$$

We then use the definition of affine transformation and substitute:

$$\|A(k_1 - k_2)\| = \|A(k(l_1 - l_2))\| = k\|A(l_1 - l_2)\|$$

Thus, the ratio is preserved even under affine transformation. However, this is not the case for non-parallel line segments.

For the case of non-parallel line segments, consider the x-y plane and the affine transformation that simply stretches by a factor of 2 in the x-direction. We see that a line segment from the origin to $(0,1)$ remains the same, but the line segment from the origin to $(1,0)$ is stretched to a length of 2. We arrive at a counterexample, and have shown that the ratio between non-parallel line segments is not invariant.

(d) No. Projective transformations map lines to lines, but do not necessarily preserve parallelism. Since area is not preserved under affine transformation (scaling), it is not preserved under projective transformations. Finally, ratios of any line segments are not invariant under projective transformations, as lines can be skewed.

## 2 Affine Camera Calibration (20 points)

In this question, we will perform affine camera calibration using two different images of a calibration grid. First, you will find correspondences between the corners of the calibration grids and the 3D scene coordinates. Next, you will solve for the camera parameters.

It was shown in class that a perspective camera can be modeled using a $3 \times 4$ matrix:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

which means that the image at point $(X, Y, Z)$ in the scene has pixel coordinates $(x/w, y/w)$. The $3 \times 4$ matrix can be factorized into intrinsic and extrinsic parameters.

An *affine* camera is a special case of this model in which rays joining a point in the scene to its projection on the image plane are parallel. Examples of affine cameras include orthographic projection and weakly perspective projection. An affine camera can be modeled as:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

which gives the relation between a scene point $(X, Y, Z)$ and its image $(x, y)$. The difference is that the bottom row of the matrix is $[\ 0 \quad 0 \quad 0 \quad 1\ ]$, so there are fewer parameters we need to calibrate. More importantly, there is no division required (the homogeneous coordinate is 1) which means this is a *linear model*. This makes the affine model much simpler to work with mathematically - at the cost of losing some accuracy. The affine model is used as an approximation of the perspective model when the loss of accuracy can be tolerated, or to reduce the number of parameters being modelled.

Calibration of an affine camera involves estimating the 8 unknown entries of the matrix in Eq. 2. (This matrix can also be factorized into intrinsics and extrinsics, but that is outside the scope of this homework). Factorization is accomplished by having the camera observe a calibration pattern with easy-to-detect corners.

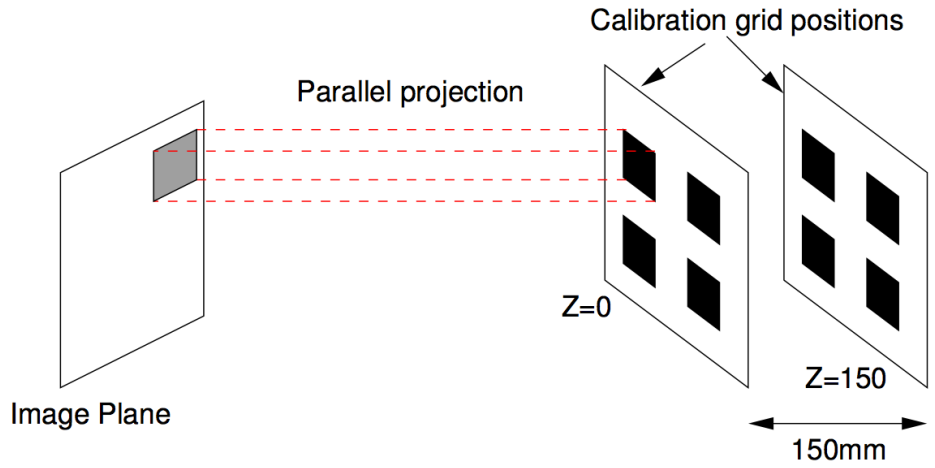
**Scene Coordinate System**

The calibration pattern used is shown in Fig. 1, which has a $6 \times 6$ grid of squares. Each square is $50mm \times 50mm$. The separation between adjacent squares is $30mm$, so the entire grid is $450mm \times 450mm$. For calibration, images of the pattern at two different positions were captured. These images are shown in Fig. 1 and can be downloaded from the course website. For the second image, the calibration pattern has been moved back (along its normal) from the rest position by $150mm$.

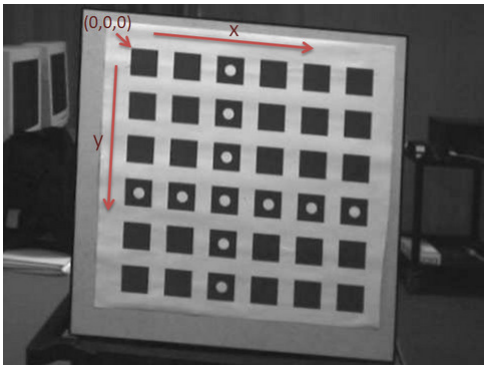
We will choose the origin of our 3D coordinate system to be the top left corner of the calibration pattern in the rest position. The $X$-axis runs left to right parallel to the rows of squares. The $Y$-axis runs top to bottom parallel to the columns of squares. We will work in units of millimeters. All the square corners from the first position corresponds to $Z = 0$. The second position of the calibration grid corresponds to $Z = 150$. The top left corner in the first image has 3D scene coordinates $(0, 0, 0)$ and the bottom right corner in the second image has 3D scene coordinates $(450, 450, 150)$. This scene coordinate system is labeled in Fig. 1.

(a) Download the calibration images from the class website (http://www.stanford.edu/class/cs231a/hw/PS1_data.zip). The images will be in the `affineCamera` directory. Find the feature correspondences manually for calibration. In other words, first calculate the scene coordinates of the square corners of the calibration grid. Next, find the corresponding pixel coordinates in the images manually.

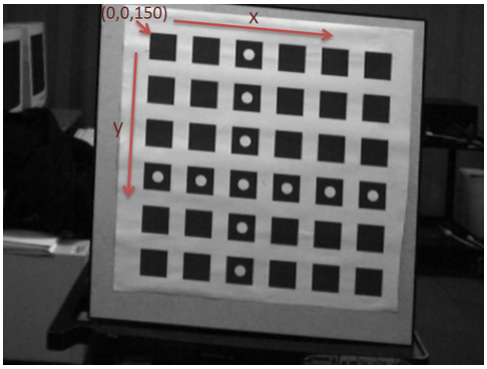
If you wish to measure the pixel coordinates interactively in MATLAB, you may find the function `ginput` useful. It is strongly recommended that you save this matrix of




(a) Image formation in an affine camera. Points are projected via parallel rays onto the image plane




(b) Image of calibration grid at Z=0




(c) Image of calibration grid at Z=150

Figure 1: Affine camera: image formation and calibration images.

measurements in a file for later use. You do not need to find *all* the corners, but the more corners you use the more accurate the calibration is likely to be. Be sure to include corners from *both* images. We recommend using at least 12 corners from each image. Report your feature correspondence pairs. You can find some helpful code for selecting points interactively in `ginputSample.m` file in the `affineCamera` directory.

(b) Having measured enough features, solve for the camera parameters using Eq. 2. Note that each measurement $(x_i, y_i) \leftrightarrow (X_i, Y_i, Z_i)$ yields two linear equations for the 8 unknown camera parameters. Given $N$ corner measurements, we have $2N$ equations and 8 unknowns. Using your measured feature correspondences as inputs, write a script which constructs the linear system of equations and solves for the camera parameters which minimizes the least-squares error. The inputs should be your feature correspondences. The output should be your calibrated camera matrix, and the RMS error between your $N$ image corner coordinates and $N$ corresponding calculated corner locations (calculated using the affine camera matrix and scene coordinates for corners). Please hand in your MATLAB code. Also, please report your $3 \times 4$ calibrated camera matrix as well as your

RMS error.

$$RMS_{total} = \sqrt{\sum((x - x')^2 + (y - y')^2)/N}$$

(c) Could you calibrate the matrix with only one checkerboard image? Explain in words.

**Solution:**

(a) No - you need to calibrate the camera on at least two planar surfaces. If only one planar surface is used, the linear system becomes rank deficient and a unique solution cannot be obtained.

(b) (c) By expanding out the original linear system, we can reformulate the question by placing all the affine camera matrix element unknowns into a single x vector (the A matrix and b vector will be known), then use least squares to find a best fit solution.
This is the original linear system:

$$x = \begin{bmatrix} x_1 & ... & x_n \\ y_1 & ... & y_n \end{bmatrix}, P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix}, X = \begin{bmatrix} X_1 & ... & X_n \\ Y_1 & ... & Y_n \\ Z_1 & ... & Z_n \\ 1_1 & ... & 1_n \end{bmatrix} \tag{3}$$

$$x = PX \tag{4}$$

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

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

$$y_1 = p_{21}X_2 + p_{22}Y_2 + p_{23}Z_2 + p_{24}$$

A similar expansion can be obtained for all the 3D world to 2D image point correspondences. Since $p_{ij}$ are the unknowns in this case, and everything else is known, we can place the $p_{ij}$ coefficients in an $x$ vector, and write out the corresponding $A$ matrix and $b$ vector of the linear system. Thus, we obtain

$$Ax = b$$

$$A = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 \\ ... & & & & & & & \end{bmatrix}$$

$$b = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ ... \end{bmatrix}$$

$$x = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \end{bmatrix}$$

Solving via least squares,

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

Alternatively, we can also solve for P directly via least norm:

$$P = xX(XX^T)^{-1} \tag{5}$$

An RMS value is obtained by projecting the 3D world coordinates onto the 2D image using the $P$ matrix we have just solved for. Next, the Euclidean distance between the projected points and the actual point on the image is obtained using the following definition for RMS:

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

where $N$ signifies the number of point correspondences used.

**Code:**

```
%CS231A Problem Set 2, Homework 4
% 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

%% Load in images
img1 = imread('fig1imgAlabel.jpg');
img2 = imread('fig1imgBlabel.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]' [80 0 0]' [160 0 0]' [240 0 0]' [320 0 0]' [400 0 0]' ...
[400 80 0]' [400 160 0]' [400 240 0]' [400 320 0]' [400 400 0]' [450 450 0]' ];
A = [ A [ 0 0 150]' [80 0 150]' [160 0 150]' [240 0 150]' [320 0 150]' [400 0 150]' ...
[400 80 150]' [400 160 150]' [400 240 150]' [400 320 150]' [400 400 150]' [450 450 150]'
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 =
%
%     0.9706




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

```
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];
```

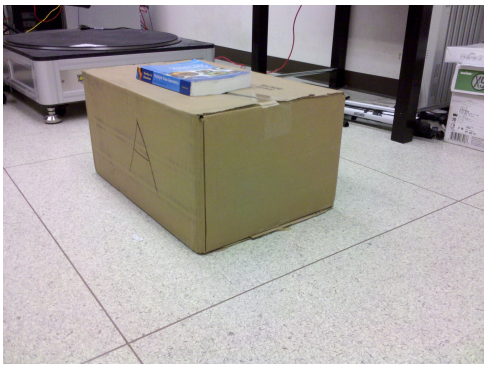
```
end

%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
```

The answer should be within the range of the following:

$$RMS = .97$$

$$P = \begin{bmatrix} 0.535 & -0.023 & 0.121 & 128.6 \\ 0.046 & 0.538 & -0.104 & 43.39 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{bmatrix}$$

# 3 Single View Geometry (30 points)



(a) Image 1 (1.jpg)

(b) Image 2 (2.jpg)

In this question, we will estimate camera parameters from a single view and leverage the projective nature of cameras to find both the camera center and focal length from vanishing points present in the scene above.

(a) Download the calibration images from the class website (http://www.stanford.edu/class/cs231a/hw/PS1_data.zip). The images will be in the `singleView` directory. Identify a set of N vanishing points from the Image 1. Assume the camera has zero skew and square pixels, with no distortion.

(i) Compute the focal length and the camera center using these N vanishing points. Show derivation and report numerical results.
*Hint: Use the function getVanishingPoint.m to compute the Vanishing Point. Read the ReadMe.txt file for usage information.*

(ii) Is it possible to compute the focal length for any choice of vanishing points?

(iii) Whats the minimum N for solving this problem? Justify answer.

(iv) The method used to obtain vanishing points (as explained in the ReadMe.txt) is approximate and prone to noise. Discuss approaches to refine this process.

This process gives the camera internal matrix under the specified constraints. For the remainder of the computations, use the following internal camera matrix:

$$K = \begin{bmatrix} 2448 & 0 & 1253 \\ 0 & 2438 & 986 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Identify a sufficient set of vanishing lines on the ground plane and the plane on which the letter A exists, written on the side of the cardboard box, (plane-A). Use these vanishing lines to verify numerically that the ground plane is orthogonal to the plane-A.

(c ) Assume the camera rotates but no translation takes place. Assume the internal camera parameters remain unchanged. An Image 2 of the same scene is taken. Use vanishing points to compute camera rotation matrix. Report derivation and numerical results.

**Solution:**

(a) (i) The focal length and camera center can be found through the camera matrix $K$ which can be derived from 3 orthogonal vanishing points, assuming the camera has zero skew and square pixels. We will find $K$ by first finding the image of the absolute conic $\omega = \left(KK^T\right)$ and later solving for $K$

Using our assumptions, we know $\omega$ has the following form:

$$\omega = \begin{bmatrix} w_1 & 0 & w_2 \\ 0 & w_1 & w_3 \\ w_2 & w_3 & w_4 \end{bmatrix}$$

The three vanishing points can be formed into three pairs of vanishing points, each pair $v_i, v_j$ provides a constraint in the form of $v_i^T \omega v_j = 0$

Knowing the structure of $\omega$, we can rearrange this constraint as follows

$$\begin{bmatrix} v_i^1 & v_i^2 & 1 \end{bmatrix} \begin{bmatrix} w_1 & 0 & w_2 \\ 0 & w_1 & w_3 \\ w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} v_j^1 \\ v_j^2 \\ 1 \end{bmatrix} = w_1 \left( v_i^1 v_j^1 + v_i^2 v_j^2 \right) + w_2 \left( v_i^1 + v_j^1 \right) + w_3 \left( v_i^2 + v_j^2 \right) + w_4$$

We can then stack these constraints and reformulate the problem as $A\mathbf{w} = \mathbf{0}$, where $A$ is a 3x4 matrix and the unknowns of $\omega$ are in $\mathbf{w}$.

Seeing that $\mathbf{w}$ is the null vector of $A$, we can recover $K$ from the expression, $\omega = \left(KK^T\right)$ by first using a Cholesky factorization of w, and then inverting the result. The actual camera matrix returned is heavily depending on the points chosen when the vanishing points were found,

An example of a camera matrix found using this method is below

$$K = \begin{bmatrix} 2931.6 & 0 & 1658.7 \\ 0 & 2931.6 & 1142.4 \\ 0 & 0 & 1.2 \end{bmatrix}$$

From this camera matrix, the estimated focal length, 2931.6, and the estimated camera center, (1658.7, 1142.4) can be found.

**Code:**

```
%Vanishing points used
v1= [-740.7300 -101.1201];
v2= [4587.1 31.7612];
v3= [1220.1 6838.5];
A=zeros(3,4);

%Calculating a row of A from each pair of vanishing points
%1,2
vi=v1;
vj=v2;
A(1,:)=[(vi(1)*vj(1) + vi(2)*vj(2)), (vi(1)+vj(1)), (vi(2)+vj(2)), (1)];

%2,3
vi=v2;
vj=v3;
A(2,:)=[(vi(1)*vj(1) + vi(2)*vj(2)), (vi(1)+vj(1)), (vi(2)+vj(2)), (1)];

%1,3
vi=v1;
vj=v3;
A(3,:)=[(vi(1)*vj(1) + vi(2)*vj(2)), (vi(1)+vj(1)), (vi(2)+vj(2)), (1)];

w=null(A);
```

```
omega=[w(1) 0 w(2)
       0 w(1) w(3)
       w(2) w(3) w(4)];

%Finding K from omega
kinv=chol(omega);
K=inv(kinv);
```

(ii) No, you need orthogonal vanishing points to get the focal length. Vanishing points also must not be at infinity.

(iii) The minimum $N$ for this problem is 3, since three points yields three distinct pairs of vanishing points for our constraints. Two points only yields 1 pair, which would make the problem under constrained.

(iv) There are many ways to refine the vanishing point positions, one way us to use more than two parallel lines and compute the a set of intersections of these lines. We can then use the average of the location of the intersections as a better estimate of the vanishing point.

(b) To determine of the two plans are orthogonal, we need to first identify two sets of parallel lines on each plane to obtain two sets of vanishing points. We can then find the vanishing lines from these points,, $l = v_1 \times v_2$ where $v_1$ and $v_2$ are vanishing points corresponding to one plane. From these two vanishing lines, we can compete the angle $\theta$ between the planes from the following expression:

$$cos\theta = \frac{\mathbf{l}_1^T \omega^* \mathbf{l}_2}{\sqrt{\mathbf{l}_1^T \omega^* \mathbf{l}_1}\sqrt{\mathbf{l}_2^T \omega^* \mathbf{l}_2}}$$

where $\omega^* = \omega^{-1} = KK^T$

We should find that the angle is approximately 90 degrees.

**Code:**

```
%Using the camera matrix specific in the question
kNew=[2448 0 1253
    0 2438 986
    0 0 1];
omegaNewStar=kNew*kNew';

%set of vanishing points in the A plane
A1=[1171.1 5939.5 1]';
A2=[-794.6047 -114.8184 1]';

%set of vanishing points in the ground plane
G1 = [1.0e+03 *[5.1711   -0.0989] 1]';
G2 = [-714.2817 -145.3947 1]';
```

```
%Finding the two vanishing lines
L1=cross(A1, A2);
L2=cross(G1, G2);

%Finding the angle between planes
costheta=(L1'*omegaNewStar*L2)/(sqrt((L1'*omegaNewStar*L1))*sqrt((L2'*omegaNewStar*L2)))

theta=acos(costheta)
```

(c) To calculate the camera rotation from vanishing points, we first have to identify corresponding vanishing points from both images, $(v_1, v_2, v_3)$ and $(v'_1, v'_2, v'_3)$ respectively.

Using the camera matrix, we can then calculate the directions of these vanishing points as measured in the camera's Euclidian coordinate frame, $d_i = \frac{K^{-1}v_i}{\|K^{-1}v_i\|}$.

The directions of the vanishing points between the first and second image are related by the following expression: $d'_i = Rd_i$ where $d'_i$ is the direction in the second image, $d_i$ is the direction in the first image, and R is the camera rotation matrix

This relationship between the directions of the vanishing points and be extended to include all 3 vanishing points

$$\left[d'_1 d'_2 d'_3\right] = R\left[d_1 d_2 d_3\right]$$

which can then be rearranged to find the camera rotation matrix as follows:

$$R = \left[d'_1 d'_2 d'_3\right]\left([d_1 d_2 d_3]\right)^{-1}$$

The output from an example solution as well as the code is below:

$$R = \begin{bmatrix} 1.0014 & 0.0399 & -0.1386 \\ -0.0369 & 1.0057 & 0.0272 \\ 0.1372 & -0.0321 & 0.9717 \end{bmatrix}$$

**Code:**

```
%first image vanishing points
v1= [-740.7300 -101.1201 1]';
v2= [4587.1 31.7612 1]';
v3= [1220.1 6838.5 1]';

%second image vanishing points
p1=[1.0e+03 *[ -1.4688   -0.1049] 1]';
p2=[1.0e+03 *[3.7815    0.1186] 1]';
p3=[1.0e+03 * [1.0988    7.6543] 1]';

%first image vanishing points directions
d1=inv(kNew)*v1/(norm(inv(kNew)*v1));
```

```
d2=inv(kNew)*v2/(norm(inv(kNew)*v2));
d3=inv(kNew)*v3/(norm(inv(kNew)*v3));

%second image vanishing points directions
dPrime1=inv(kNew)*p1/(norm(inv(kNew)*p1));
dPrime2=inv(kNew)*p2/(norm(inv(kNew)*p2));
dPrime3=inv(kNew)*p3/(norm(inv(kNew)*p3));

dOld=[d1, d2, d3];
dNew=[dPrime1, dPrime2, dPrime3];
R=dNew*dOld^(-1)
```

# 4 Efficient Solution for Stereo Correspondence(25 points)

In this exercise you are given two or more images of the same 3D scene, taken from different points of view. You will be asked to solve a correspondence problem to find a set of points in one image which can be identified as the same points in another image. Specifically, the way you formulate the correspondence problem in this exercise will lend itself to be computationally efficient.

(a) Suppose you are given a point $(p_1, p_2)$ in the first image and the corresponding point $(p_1', p_2')$ in the second image, both of the same 3D scene. Write down the homogenous coordinates $x$ and $x'$. In addition, write down the necessary conditions for the points $x$ and $x'$ to meet in the 3D world, using what we know about fundamental matrix $F$.

(b) Since our points are measurements and susceptible to noise, $x$ and $x'$ may not satisfy the conditions in (a). Writing this statistically, our measurements are given by

$$x = \bar{x} + \Delta x \qquad x' = \bar{x}' + \Delta x'$$

where $\Delta x$ and $\Delta x'$ are noise terms, and $\bar{x}$ and $\bar{x}'$ are the true correspondences we are trying to determine. Now we require that only $\bar{x}$ and $\bar{x}'$ satisfy the condition in (a). To find our best estimate of $\bar{x}$ and $\bar{x}'$, we will minimize

$$E = \|\Delta x\|^2 + \|\Delta x'\|^2$$

subject to the constraint on $\bar{x}$ and $\bar{x}'$ from (a). In addition to the constraint from (a), we also need to constrain the noise terms so that $x$ and $x'$ remain on the image plane, i.e., a constraint of $\Delta x$ so that the last coordinate of $x$ and $\bar{x}$ in homogeneous coordinate is identical. Similar for $x'$ and $\bar{x}'$. For this part, write down the optimization problem (i.e. what you are minimizing and the constraints). Disregard the higher order terms of $\Delta x$ and $\Delta x'$ in the constraint from (a).

Your answer should be in the following form:

minimize _____

subject to _____

_____

_____

*Hint: To constrain the noise terms $\Delta x$ and $\Delta x'$ to lie on the image plane, the unit vector $e_3 = [\ 0 \quad 0 \quad 1\ ]^T$ will be useful. Also, $\bar{x}$ and $\bar{x}'$ should not appear in the optimization problem.*

(c) Once we drop these higher order terms, we can use Lagrange multipliers to solve the optimization problem in (b). Show that optimal $\Delta x$ and $\Delta x'$ are given by

$$\Delta x = \frac{x^T F x' P_e F x'}{x'^T F^T P_e F x' + x^T F P_e F^T x} \qquad \Delta x' = \frac{x^T F x' P_e F^T x}{x'^T F^T P_e F x' + x^T F P_e F^T x}$$

*Hint: The projection matrix $P_e = \text{diag}(1, 1, 0)$ will be useful to eliminate unwanted terms after you have taken the derivative and set it to zero.*

*Remark: Once we have determined the optimal values for $\Delta x$ and $\Delta x'$ we can use these optimal values in conjunction with our measurements $x$ and $x'$ to determine our estimate of the true correspondences $\bar{x}$ and $\bar{x}'$.*

**Solution:**

(a) Writing the homogenous coordinates we have

$$x = \begin{bmatrix} p_1/f_0 \\ p_2/f_0 \\ 1 \end{bmatrix} \qquad x' = \begin{bmatrix} p_1'/f_0 \\ p_2'/f_0 \\ 1 \end{bmatrix}$$

From the lecture, the necessary and sufficient conditions for the line of sight points to map to the same world coordinates is the epipolar equation $x^T F x' = 0$.

(b) We begin by rewriting $\bar{x}$ and $\bar{x}'$ as

$$\bar{x} = x - \Delta x \qquad \bar{x}' = x' - \Delta x'$$

Using the condition from part (a) on $\bar{x}$ and $\bar{x}'$ we have

$$(x - \Delta x)^T F (x' - \Delta x') = 0$$

Expanding and ignoring higher order terms we find

$$x'^T F^T \Delta x + x^T F \Delta x' = x^T F x'$$

. For points $x$ and $x'$ to remain on the image plane the noise terms must satisfy

$$e_3^T \Delta x = 0 \qquad e_3^T \Delta x' = 0$$

Thus our optimization problem becomes

$$\begin{aligned} \text{minimize} \quad & \|\Delta x\|^2 + \|\Delta x'\|^2 \\ \text{subject to} \quad & x'^T F^T \Delta x + x^T F \Delta x' = x^T F x' \\ & e_3^T \Delta x = 0 \\ & e_3^T \Delta x' = 0 \end{aligned}$$

(c) Using Lagrange multipliers we can write out our Lagrangian for the optimization problem we wrote down in part (b) as

$$\|\Delta x\|^2 + \|\Delta x'\|^2 - \lambda \left( x'^T F^T \Delta x + x^T F \Delta x' \right) - \mu_1 e_3^T \Delta x - \mu_2 e_3^T \Delta x'$$

finding the derivatives with respect to $\Delta x$ and $\Delta x'$ and setting them to zero we have

$$2\Delta x - \lambda F x' - \mu_1 e_3 = \mathbf{0} \quad 2\Delta x' - \lambda F^T x - \mu_2 e_3 = \mathbf{0}$$

Using the hint we multiply both equations by the projection matrix $P_e = \text{diag}(1, 1, 0)$ we have $P_e \Delta x = \Delta x$, $P_e \Delta x' = \Delta x'$ and $P_e e_3 = \mathbf{0}$ thus we can write

$$2\Delta x - \lambda P_e F x' = \mathbf{0} \quad 2\Delta x' - \lambda P_e F^T x = \mathbf{0}$$

Solving we find

$$\Delta x = \frac{\lambda}{2} P_e F x' \quad \Delta x' = \frac{\lambda}{2} P_e F^T x$$

Then substituting this into the answer to part (b) we have

$$\frac{\lambda}{2} x'^T F^T P_e F x' + \frac{\lambda}{2} x^T F P_e F^T x = x^T F x'$$

solving for $\lambda$ we have

$$\frac{\lambda}{2} = \frac{x^T F x'}{x'^T F^T P_e F x' + x^T F P_e F^T x}$$

substituting this value of $\lambda$ in to the equations we found for $\Delta x$ and $\Delta x'$ we can write

$$\Delta x = \frac{x^T F x' P_e F x'}{x'^T F^T P_e F x' + x^T F P_e F^T x}$$
$$\Delta x' = \frac{x^T F x' P_e F^T x}{x'^T F^T P_e F x' + x^T F P_e F^T x}$$