# CS 231A Computer Vision (Fall 2011)
# Problem Set 2

**Solution Set**

Due: Oct. 28$^{\text{th}}$, 2011 (5pm)

## 1 Some Projective Geometry Problems (15 points)

Suppose there are two parallel lines that extend to infinity in our world coordinates. The images of these two lines intersect at a point, $v$, on the image plane known as the vanishing point. In this problem we choose to represent the lines as

$$l = \left\{ r \begin{bmatrix} \cos\alpha \\ \cos\beta \\ \cos\gamma \end{bmatrix} \;\middle|\; r \in \mathbb{R} \right\}$$

where $\alpha$, $\beta$ and $\gamma$ are the angles from the $x$, $y$ and $z$ world axes respectively.

(a) Using homogenous coordinates to show that the image coordinates of the vanishing point $v$ can be written as $v = KRd$ where

$$d = \begin{bmatrix} \cos\alpha \\ \cos\beta \\ \cos\gamma \end{bmatrix}$$

$K$ is the camera calibration matrix and $R$ is the rotation matrix between the camera and the world coordinates.

(b) Express the line direction $d$ in terms of $K$,$R$ and $v$.

*Remark : Any matrix inverses which do not hold in general, must be shown to hold in this case.*

(c) Now consider the situation where three lines intersect such that each line is orthogonal to the other two. That is, each line's directional vector $d$ satisfies the following

$$d_1^T d_2 = 0 \quad d_1^T d_3 = 0 \quad d_2^T d_3 = 0$$

Using this property, show that

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

for $i \neq j$

**Solution:**

(a) As mentioned in the problem a group of parallel lines intersect at one point on the image plane as the world coordinate radius from the image plane goes to infinity. Without loss of generality, we consider only one of the lines, denoted by $l_0$, that goes through the origin of world coordinate system. Using the representation of the line given in the problem we can write the homogenous coordinate representation of our line as

$$l = \left\{ \begin{bmatrix} \cos\alpha \\ \cos\beta \\ \cos\gamma \\ 1/r \end{bmatrix} \;\middle|\; r \in \mathbb{R} \right\}$$

letting r $\to \infty$ the image of the line $l_0$ with reach the vanishing point. Therefore using the equations from lecture to convert between world coordinates and image coordinates we can write the vanishing point as

$$v = \lim_{r\to\infty} \begin{bmatrix} K & \mathbf{0} \end{bmatrix} \begin{bmatrix} R & T \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \cos\alpha \\ \cos\beta \\ \cos\gamma \\ 1/r \end{bmatrix}$$

$$= \begin{bmatrix} K & \mathbf{0} \end{bmatrix} \begin{bmatrix} R & T \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \cos\alpha \\ \cos\beta \\ \cos\gamma \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} K & \mathbf{0} \end{bmatrix} \begin{bmatrix} Rd \\ \mathbf{0} \end{bmatrix}$$
$$= KRd$$

where

$$d = \begin{bmatrix} \cos\alpha \\ \cos\beta \\ \cos\gamma \end{bmatrix}$$

(b) In lecture we have already shown $K$ is of the form:

$$K = \begin{bmatrix} \alpha & -\alpha\cot\theta & u_0 \\ 0 & \frac{\beta}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

since $|K| = \alpha\beta/\sin\theta \neq 0$, $K^{-1}$exists. Here we use the fact that any rotation can be decomposed into three single rotations around each axis, i.e.

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

now considering $|R_i| = \cos^2\theta_i + \sin^2\theta_i = 1$ and $|R| = |R_x R_y R_z| = |R_x||R_y||R_z| = 1$, $R^{-1}$ exists. Therefore we can write $d = R^{-1}K^{-1}v$.

(c) From the last problem we have $d = R^{-1}K^{-1}v \Rightarrow Rd = K^{-1}v$. Also we make use of the fact that any rotation matrix is a unitary matrix, i.e. $R^T R = I$.

$$\begin{aligned} d_i^T d_j &= 0 \\ d_i^T (R^T R) d_j &= 0 \\ (Rd_i)^T (Rd_j) &= 0 \\ (K^{-1}v_i)^T (K^{-1}v_j) &= 0 \end{aligned}$$

This holds for $i \neq j$

## 2 Fundamental Matrix (15 points)

In this question, you will explore some properties of fundamental matrix and derive a minimal parameterization for it.

(a) Show that two $3 \times 4$ camera matrices $M$ and $M'$ can always be reduced to the following canonical forms by an appropriate projective transformation in a 3D space, which is represented by a $4 \times 4$ matrix $H$. Here, we assume $e_3^T(-A'A^{-1}b + b') \neq 0$, where $e_3 = (0, 0, 1)^T$, $M = [A, b]$ and $M' = [A', b']$.
Note: You don't have to show the explicit form of $H$ for the proof.
*(Hint: The camera matrix has rank 3.)*

$$\hat{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ and } \hat{M}' = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) Given a $4 \times 4$ matrix $H$ representing a projective transformation in a 3D space, prove that the fundamental matrices corresponding to the two pairs of camera matrices $(M, M')$ and $(MH, M'H)$ are the same.

(c) Using the conclusions from (a) and (b), derive the fundamental matrix $F$ of the camera pair $(M, M')$ using $a_{11}$, $a_{12}$, $a_{13}$, $a_{21}$, $a_{22}$, $a_{23}$, $b_1$, $b_2$. Then use the fact that $F$ is only defined up to a scale factor to construct a seven-parameter expression for $F$. *(Hint: The fundamental matrix corresponding to a pair of camera matrices $M = [I \mid 0]$ and $M' = [A \mid b]$ is equal to $[b]_\times A$.)*

**Solution:**

(a) Since the camera matrix $M$ has rank 3, we can always first find $4 \times 4$ matrix $H_0$

$$H_0 = \begin{bmatrix} A^{-1} & -A^{-1}b \\ 0 & 1 \end{bmatrix}.$$

such that

$$MH_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Applying the same transformation to $M'$ we will get a new matrix,

$$M'H_0 = [A'A^{-1}, -A'A^{-1}b + b'] = \begin{bmatrix} m'_{11} & m'_{12} & m'_{13} & m'_{14} \\ m'_{21} & m'_{22} & m'_{23} & m'_{24} \\ m'_{31} & m'_{32} & m'_{33} & m'_{34} \end{bmatrix}.$$

As $e_3^T(-A'A^{-1}b + b') \neq 0$, $m'_{34} \neq 0$.

Now multiply another matrix

$$H_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\frac{m'_{31}}{m'_{34}} & -\frac{m'_{32}}{m'_{34}} & -\frac{m'_{33}}{m'_{34}} & \frac{1}{m'_{34}} \end{bmatrix}$$

to the right of both of them, we will have

$$MH_0H_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ and } MH_0H_1 = \begin{bmatrix} m'_{11} - \frac{m'_{14}m'_{31}}{m'_{34}} & m'_{12} - \frac{m'_{14}m'_{32}}{m'_{34}} & m'_{13} - \frac{m'_{14}m'_{33}}{m'_{34}} & \frac{m'_{14}}{m'_{34}} \\ m'_{21} - \frac{m'_{24}m'_{31}}{m'_{34}} & m'_{22} - \frac{m'_{24}m'_{32}}{m'_{34}} & m'_{23} - \frac{m'_{24}m'_{33}}{m'_{34}} & \frac{m'_{24}}{m'_{34}} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

So a projection transformation matrix $H = H_0H_1$ will reduce $M$ and $M'$ to the canonical forms.

(b) Observe that $MX = (MH)(H^{-1}\mathbf{X})$, and similarly for $M'$. Thus if $\mathbf{x}$ and $\mathbf{x}'$ are matched points with respect to the pair of cameras $(M, M')$, corresponding to a 3D point $\mathbf{X}$, then they are also matched points with respect to the pair of cameras $(MH, M'H)$, corresponding to the point $H^{-1}\mathbf{X}$.

(c) From the conclusion of (b), the fundamental matrix of the camera pair $(M, M')$ is the same as the fundamental matrix of the camera pair $(\hat{M}, \hat{M}')$, which is $[\hat{b}]_\times \hat{A}$ where

$$\hat{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 \end{bmatrix}, \hat{b} = \begin{bmatrix} b_1 \\ b_2 \\ 1 \end{bmatrix}$$

Thus $F$ can be found.

$$F = [\hat{b}]_\times \hat{A} = \begin{bmatrix} 0 & -1 & b_2 \\ 1 & 0 & -b_1 \\ -b_2 & b_1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -a_{21} & -a_{22} & -a_{23} \\ a_{11} & a_{12} & a_{13} \\ -a_{11}b_2 + a_{21}b_1 & -a_{12}b_2 + a_{22}b_1 & -a_{13}b_2 + a_{23}b_1 \end{bmatrix}$$

We can divide $F$ by $a_{21}$ to get a seven-parameter expression,

$$\begin{bmatrix} -1 & -\frac{a_{22}}{a_{21}} & -\frac{a_{23}}{a_{21}} \\ \frac{a_{11}}{a_{21}} & \frac{a_{12}}{a_{21}} & \frac{a_{13}}{a_{21}} \\ -\frac{a_{11}}{a_{21}}b_2 + b_1 & -\frac{a_{12}}{a_{21}}b_2 + \frac{a_{22}}{a_{21}}b_1 & -\frac{a_{13}}{a_{21}}b_2 + \frac{a_{23}}{a_{21}}b_1 \end{bmatrix}.$$

The new seven parameters are $\frac{a_{11}}{a_{21}}$, $\frac{a_{12}}{a_{21}}$, $\frac{a_{13}}{a_{21}}$, $\frac{a_{22}}{a_{21}}$, $\frac{a_{23}}{a_{21}}$, $b_1$, $b_2$.

## 3 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 and sufficient conditions for the points $x$ and $x'$ to meet in the 3D world in terms of 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. 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 $P_e = \mathrm{diag}(1, 1, 0)$ will be useful to eliminate unwanted terms*

*after you 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

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

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

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

# 4 Affine Camera Calibration (20 points)

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 0 0 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 is 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.

**Corner Extraction**

*The task that you must complete for this question begins here.* Extract the feature correspondences required for calibration. In other words, first find the scene coordinates of the square corners of the calibration grid. Next, find the corresponding pixel coordinates in the images.

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 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.

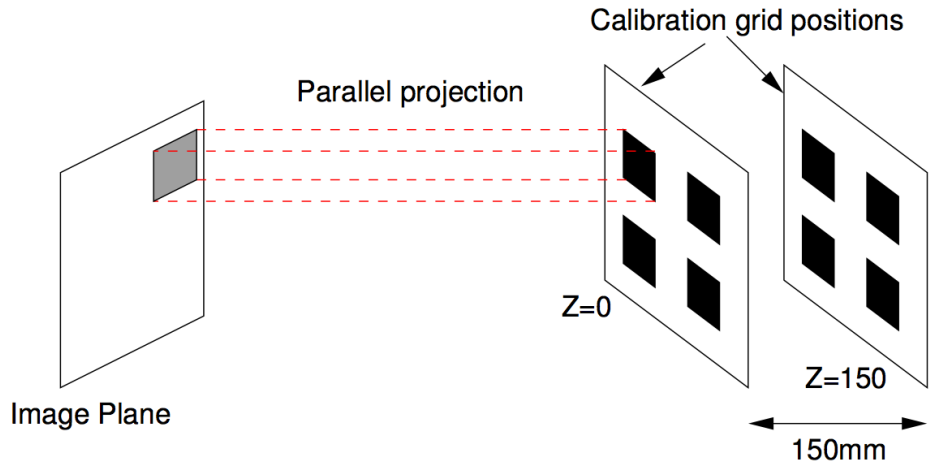
**Solving for the camera matrix**

Having measured enough features, we can now 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 input, write code which constructs the linear system of equations and solves for the camera parameters which minimizes the least-squares error.

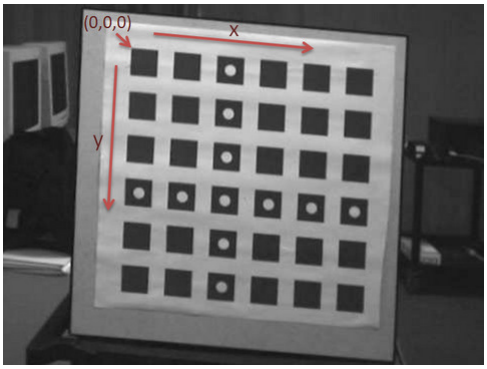
(a) Could you calibrate the matrix with only one checkerboard image? Explain why.

(b) The RMS error (in pixels) between the corner positions and the positions predicted by the camera parameters.

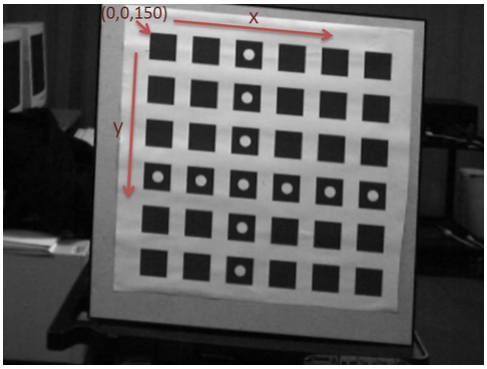
(c) Give your $3 \times 4$ calibrated camera matrix.




(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.

**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:

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

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

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

$$Ax = b$$

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

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

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

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:

$$RMS = .97$$

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

# 5 Image Stitching (25 points)

In this problem, you will implement your own image stitching application. This technique is used to combine many photographs to produce a larger image than could be possible with one exposure alone. Refer to Fig. 2 for an example output from Adobe Photoshop's photomerge capability which produces highly refined panoramas using advanced techniques such as feathering, pyramid blending. Ignoring the "bells and whistles," the image-stitching portion of this algorithm reduces to finding the parameters for the affine transformation and homography matrix $H$ to map one image onto the plane of the other.

(a) Let $p$ correspond to a point on one image and let $p'$ correspond to the same point in the scene, but projected onto another image. Write a general equation for how a homography matrix $H$ maps points from one image to another. How would H be restricted if it must describe an affine transformation?

(b) Our first implementation of image-stitching will use an affine approximation. This matrix transforms any point $p_i$ in one view to its corresponding homogeneous coordinates in the second view, $p_i'$, in accordance with your equation above. Note that $p_i, p_i' \in \mathbb{R}^3$.

*Your task is to write a function that takes in $n \geq 4$ pairs of corresponding points from the two views, where each point is specified with its 2-D image coordinates, and returns the affine transformation matrix.* Note: We have provided skeleton code. Please download `PS2_data.zip` from the course webpage. The skeleton code `stitching.m` and `findHAMatrix.m` can be found in the `imageStitching` directory. You can confirm your result with MATLAB's `cp2tform` command which is given in `stitching.m`. Please hand in the result stitched image, and your code in `findHAMatrix.m`.

*Big Hint: We can set up a solution using a system of linear equations $Ax = b$, where the 8 unknowns of $H$ are put into a vector $x \in \mathbb{R}^8$, $b \in \mathbb{R}^{2n}$ contains image points from one view and the $2n \times 8$ matrix $A$ is filled appropriately so that the full system gives us $\lambda p_i = Hp_i'$. Be sure to apply what you know about H. Solve for the unknown homography matrix parameters.*

(c) Write a function in `findHMatrix.m` which takes a set of corresponding image points ($n \geq 4$ pairs) and computes the associated $3 \times 3$ general homography matrix $H$. Use this function instead of your `findHAMatrix` function from part (b) to stitch the images together. As in (b), please hand in the result stitched image,and your code in `findHMatrix.m`.

Which method (affine transformation, or homography) produces better results?

(d) Stitch a panorama using your own photos (number of images $n \geq 4$).

Figure 2: Example panorama

**Solution:**

(a) The mapping is of the form $Hp = p'$. The affine transform is a subset of homographies where the third row of $H$ is [0 0 1]. Your linear equation for a single point correspondence takes the following form:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}$$

Note that we are using homogeneous coordinates in this case, but our point correspondences are in image coordinates. In the affine homography case, $w = 1$.

(b) A similar technique is used as in problem 4, where the unknown elements of the $H$ matrix are placed in a vector $x$. Next, the $A$ matrix and $b$ vector are determined, and we can solve for $x$ using linear least squares. This should be your setup:

$$Ax = b$$

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ \cdots & & & & & \end{bmatrix}$$

$$b = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \cdots \end{bmatrix}$$

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

Solving via least squares,

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

(c) Solving for a general homography matrix is very similar to solving for your affine transformation matrix, except now the constraint on the last row of $H$ has been relaxed. In general, a homography matrix has 9 unknowns. However, there is an inherent scale ambiguity of the homography matrix due to the use of homogeneous coordinates. Thus, to allow for a unique solution for your homography matrix, we fix $p_{33} = 1$.

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}$$

Note that we are using homogeneous coordinates in this case, but our point correspondences are in image coordinates. Here is the expanded expression for a single point correspondence.

$$p_{31}x + p_{32}y + 1 = w$$
$$p_{11}x + p_{12}y + p_{13} = wx'$$
$$p_{21}x + p_{22}y + p_{23} = wy'$$

In the general homography case, $w$ is no longer to be guaranteed to be 1 so we must write $w$ in terms of the $H$ matrix elements and take it into account when solving for them.

$$p_{11}x + p_{12}y + p_{13} = (p_{31}x + p_{32}y + 1)x'$$

$$p_{21}x + p_{22}y + p_{23} = (p_{31}x + p_{32}y + 1)y'$$

Rearranging the terms, we obtain:

$$p_{11}x + p_{12}y + p_{13} - p_{31}xx' - p_{32}yx' = x'$$

$$p_{21}x + p_{22}y + p_{23} - p_{31}xy' - p_{32}yy' = y'$$

Similar equations can be written for the remaining point correspondences. A similar technique is used as in problem 4, where the unknown elements of the $H$ matrix are placed in a vector $x$. Next, the $A$ matrix and $b$ vector are determined, and we can solve for $x$ using linear least squares. This should be your setup:

$$Ax = b$$

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ \dots & & & & & & & \end{bmatrix}$$

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

$$x = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{31} \\ p_{32} \end{bmatrix}$$

Solving via least squares,

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

The general homography method should produce better stitching results (the pixels from the 2 images should match better when stitched together) because in general, the rotation of a camera about its center is modeled by a general homography, not an affine transform. The affine transform is merely a simplifcation of the general homography to produce acceptable results, while reducing the amount of computation required to compute the transformation matrix.

(a) Image Stitching Using Affine Transform

(b) Image Stitching Using Homography

Figure 3: Sample output

**Code:**

```
%Problem 5: Image Stitching
%This code is a skeleton for you to implement yor own image stitching
%algorithm. This is not a formal structure, but should give you everything
%you need to build your own stitching algorithm.
%Running this code with the current settings will simply overlay the two
%images on top of each other

%Pick your test images.
img1_name = 'uttower2.JPG';
img2_name = 'uttower1.JPG';

img1 = imread(img1_name);
img2 = imread(img2_name);

%CPSelect will open MATLAB's Control Point Selection UI
%This will output two arrays; base_points and input_points
%You do not have to call this cpselect every time, providing you save
%the vectors input_points and base_points.
cpselect(img1_name, img2_name); pause;

%Here you calculate the transformation matrix with your code

%YOUR CODE HERE


H = findHMatrix(base_points, input_points);
H = findHAMatrix(base_points, input_points);  % this is for the
% affine case
TFORM = maketform('projective',H');

%END CODE

%The remainder of this code is just plotting your result
% test = imtransform(img1, TFORM);
% imshow(test);

[trans xdata ydata] = imtransform(img1, TFORM);
[n o b] = size(trans);
[l m z] = size(img2);

%Now combine the two images into a final output
combine_images(xdata,ydata, trans, img2)
```

```
%-------------------------------------------------------------------------------------
%returns the H matrix given point and pointp (transformed coordinates)
function out = findHMatrix (point, pointp)

% 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 1 0  0  0  -x1*x1p -y1*x1p;
%      0  0  0 x1 y1 1  -x1*y1p -y1*y1p;
%      x2 y2 1 0  0  0  -x2*x2p -y2*x2p;
%      0  0  0 x2 y2 1  -x2*y2p -y2*y2p;
%      x3 y3 1 0  0  0  -x3*x3p -y3*x3p;
%      0  0  0 x3 y3 1  -x3*y3p -y3*y3p;
%      x4 y4 1 0  0  0  -x4*x4p -y4*x4p;
%      0  0  0 x4 y4 1  -x4*y4p -y4*y4p;
%      ];
%
% 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);

    %calculates A matrix and B vector
    A = [A;
         xCurrent yCurrent 1 0 0 0 -xCurrent * xpCurrent -yCurrent * xpCurrent;
         0 0 0 xCurrent yCurrent 1  -xCurrent * ypCurrent -yCurrent * ypCurrent;];
     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 1];
 out = H;

end


%-------------------------------------------------------------------------------------
%returns the affine matrix given point and pointp (transformed coordinates)
function out = findHAMatrix (point, pointp)

% A description assuming 4 correspondences
% 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 1 0  0  0 ;
%      0  0  0 x1 y1 1 ;
%      x2 y2 1 0  0  0 ;
```

```
%      0  0  0 x2 y2 1 ;
%      x3 y3 1 0  0  0 ;
%      0  0  0 x3 y3 1 ;
%      x4 y4 1 0  0  0 ;
%      0  0  0 x4 y4 1 ;
%      ];
%
% B = [x1p y1p x2p y2p x3p y3p]';
%

%populates A matrix and b vector
%A matrix and b vector are calculated by writing out the matrix
%multiplication of the original linear algebra problem, placing all
%unknowns in the x vector, and populating the corresponding A matrix and b
%vector which should be in terms of the known points.
A = [];
b = [];
for index = 1:size(point,1)
    xCurrent = point(index,1);
    xpCurrent = pointp(index,1);
    yCurrent = point(index,2);
    ypCurrent = pointp(index,2);

    A = [A;
         xCurrent yCurrent 1 0 0 0;
         0 0 0 xCurrent yCurrent 1];
     b = [b; xpCurrent; ypCurrent];
end

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

 a = x(1);
 b = x(2);
 c = x(3);
 d = x(4);
 e = x(5);
 f = x(6);
 g = 0;
 h = 0;

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

end
```