

# PS1 Review Session

Andrey Kurenkov  
CS231A  
01/14/2022

# Problem Outline

- Q1: Projective Geometry
- Q2: Affine Camera Calibration
- Q3: Single View Geometry

# Problem Outline

- Q1: Projective Geometry
- Q2: Affine Camera Calibration
- Q3: Single View Geometry

# Problem Outline

## Projective Geometry Problems [20 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.

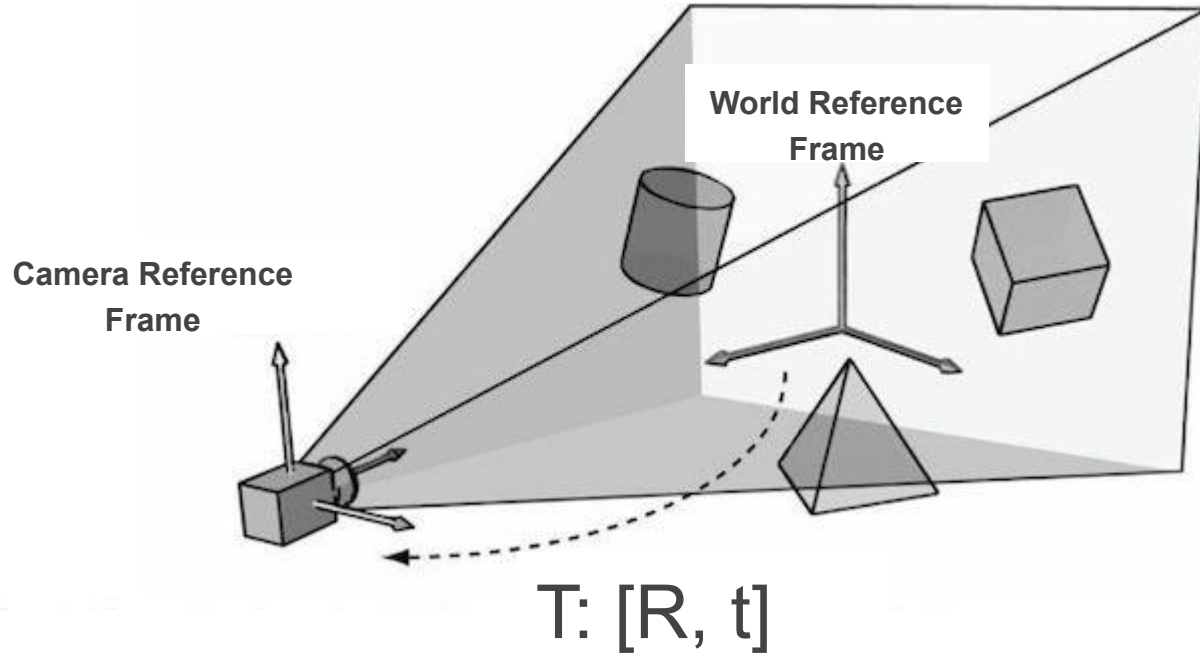
- (a) Prove that parallel lines in the world reference system are still parallel in the camera reference system. [4 points]
- (b) Consider a unit square  $pqrs$  in the world reference system where  $p, q, r$ , and  $s$  are points. Will the same square in the camera reference system always have unit area? Prove or provide a counterexample. [4 points]
- (c) 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. [6 points]

- (d) You have explored whether these three properties hold for affine transformations. Do these properties hold under any projective transformation? Justify briefly in one or two sentences (no proof needed). [6 points]


# P1: Reference Frames



# P1: Cross Products

- Lines  $k$  and  $l$  are parallel
  - $k_1$  and  $k_2$  are any two points on  $k$
  - $l_1$  and  $l_2$  are any two points on  $l$
  - by definition of parallel lines:

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


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

$$(Rk_1 - Rk_2) \times (Rl_1 - Rl_2)$$

# P1: b

- Given a square  $pqrs$ ,
  - Area =

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

- Hint: projecting to camera frame is isometric (they preserve lengths, as well as angles between lines)

# P1: 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,

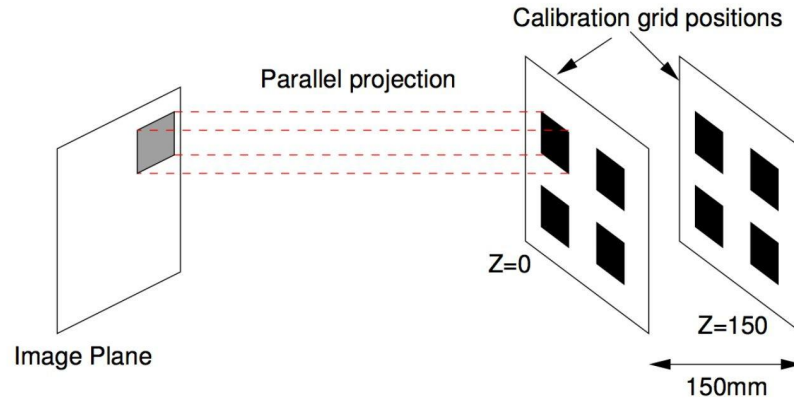
$$\|k_1 - k_2\| = k\|l_1 - l_2\|$$



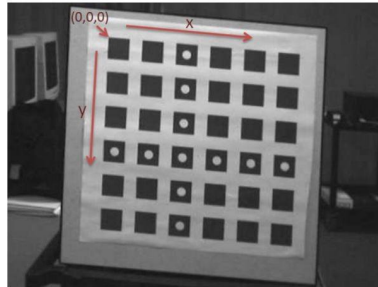
# Problem Outline

- Q1: Projective Geometry
- Q2: Affine Camera Calibration
- Q3: Single View Geometry

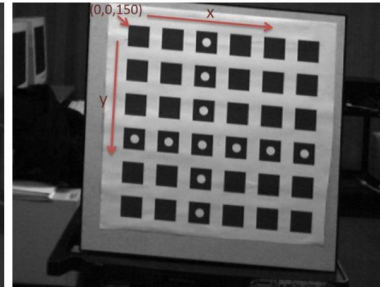
# P2: Setup



(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  $Z=150$

# P2: Perspective Camera Model

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\mathbf{P}} \quad \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} \frac{\mathbf{m}_1 \mathbf{P}_i}{\mathbf{m}_3 \mathbf{P}_i} \\ \frac{\mathbf{m}_2 \mathbf{P}_i}{\mathbf{m}_3 \mathbf{P}_i} \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{M} \mathbf{P}_w = \underbrace{\mathbf{K}}_{\text{Internal parameters}} \underbrace{[\mathbf{R} \quad \mathbf{T}]}_{\text{External parameters}} \mathbf{P}_w$$

## P2: Affine Camera Model

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

- **Linear**
- **8 Unknowns**

# Problem Outline

- (a) Given correspondences for the calibrating grid, 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 the given corner correspondences as inputs, complete the method `compute_camera_matrix()`. You will construct a linear system of equations and solve for the camera parameters to minimize the least-squares error. After doing so, you will return the  $3 \times 4$  affine camera matrix composed of these computed camera parameters. In your written report, submit your code as well as the camera matrix that you compute. **[15 points]**
- (b) After finding the calibrated camera matrix, you will compute the RMS error between the given  $N$  image corner coordinates and  $N$  corresponding calculated corner locations in `rms_error()`. Recall that

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

Please submit your code and the RMS error for the camera matrix that you found in part (a). **[15 points]**

- (c) Could you calibrate the matrix with only one checkerboard image? Explain briefly in one or two sentences. **[5 points]**

## P2: Affine Camera Model

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

$$x = PX \quad (4)$$

# P2: Affine Camera Model

$$P' = M P_w = \underbrace{K}_{\text{Internal parameters}} \underbrace{[R \ T]}_{\text{External parameters}} P_w$$

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} \frac{\mathbf{m}_1 P_i}{\mathbf{m}_3 P_i} \\ \frac{\mathbf{m}_2 P_i}{\mathbf{m}_3 P_i} \end{bmatrix}$$

$$\left\{ \begin{array}{l} u_1(\mathbf{m}_3 P_1) - \mathbf{m}_1 P_1 = 0 \\ v_1(\mathbf{m}_3 P_1) - \mathbf{m}_2 P_1 = 0 \\ \vdots \\ u_i(\mathbf{m}_3 P_i) - \mathbf{m}_1 P_i = 0 \\ v_i(\mathbf{m}_3 P_i) - \mathbf{m}_2 P_i = 0 \\ \vdots \\ u_n(\mathbf{m}_3 P_n) - \mathbf{m}_1 P_n = 0 \\ v_n(\mathbf{m}_3 P_n) - \mathbf{m}_2 P_n = 0 \end{array} \right.$$

[Eqs. 3]

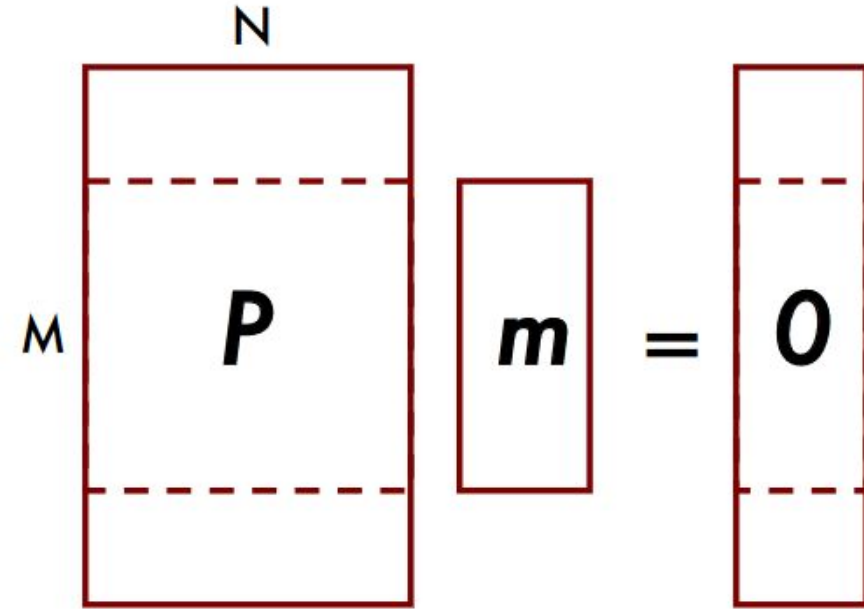
## P2: Affine Camera Model

$$\mathbf{P} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{P}_1^T & \mathbf{0}^T & -u_1 \mathbf{P}_1^T \\ \mathbf{0}^T & \mathbf{P}_1^T & -v_1 \mathbf{P}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{P}_n^T & \mathbf{0}^T & -u_n \mathbf{P}_n^T \\ \mathbf{0}^T & \mathbf{P}_n^T & -v_n \mathbf{P}_n^T \end{pmatrix} \begin{matrix} 1 \times 4 \\ 2n \times 12 \end{matrix}$$

$$\mathbf{m} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{pmatrix} \begin{matrix} 4 \times 1 \\ 12 \times 1 \end{matrix}$$



## P2: Affine Camera Model



Rectangular system ( $M > N$ )

- $\mathbf{0}$  is always a solution
- To find non-zero solution

Minimize  $\|\mathbf{P}\mathbf{m}\|^2$

under the constraint  $\|\mathbf{m}\|^2 = 1$

# P2: Affine Camera Model

## DATA FORMAT

In this problem, we provide and load the data for you. Recall that in the original problem statement, there exists a grid of black squares on a white background. We know how these black squares are setup, and thus can determine the locations of specific points on the grid (namely the corners). We also have images taken of the grid at a front image (where  $Z = 0$ ) and a back image (where  $Z = 150$ ). The data we load for you consists of three parts: `real_XY`, `front_image`, and `back_image`. For a corner  $(0,0)$ , we may see it at the  $(137, 44)$  pixel in the front image and the  $(148, 22)$  pixel in the back image. Thus, one row of `real_XY` will contain the numpy array  $[0, 0]$ , corresponding to the real XY location  $(0, 0)$ . The matching row in `front_image` will contain  $[137, 44]$  and the matching row in `back_image` will contain  $[148, 22]$

...

## COMPUTE\_CAMERA\_MATRIX

Arguments:

`real_XY` - Each row corresponds to an actual point on the 2D plane

`front_image` - Each row is the pixel location in the front image where  $Z=0$

`back_image` - Each row is the pixel location in the back image where  $Z=150$

Returns:

... `camera_matrix` - The calibrated camera matrix (3x4 matrix)

`linalg.lstsq(a, b, rcond='warn')`

[\[source\]](#)

Return the least-squares solution to a linear matrix equation.

Computes the vector  $x$  that approximately solves the equation  $a @ x = b$ . The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rows of  $a$  can be less than, equal to, or greater than its number of linearly independent columns). If  $a$  is square and of full rank, then  $x$  (but for round-off error) is the “exact” solution of the equation. Else,  $x$  minimizes the Euclidean 2-norm  $\|b - ax\|$ . If there are multiple minimizing solutions, the one with the smallest 2-norm  $\|x\|$  is returned.

Parameters:  $a : (M, N)$  *array\_like*

“Coefficient” matrix.

$b : \{(M, ), (M, K)\}$  *array\_like*

Ordinate or “dependent variable” values. If  $b$  is two-dimensional, the least-squares solution is calculated for each of the  $K$  columns of  $b$ .

# Hi  
# ai  
#  
# Oa  
# Ai  
# Lc

, 2)

# P2: Affine Camera Model

`linalg.lstsq(a, b, rcond='warn')`

[\[source\]](#)

Return the least-squares solution to a linear matrix equation.

Computes the vector  $x$  that approximately solves the equation  $a @ x = b$ . The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rows of  $a$  can be less than, equal to, or greater than its number of linearly independent columns). If  $a$  is square and of full rank, then  $x$  (but for round-off error) is the “exact” solution of the equation. Else,  $x$  minimizes the Euclidean 2-norm  $\|b - ax\|$ . If there are multiple minimizing solutions, the one with the smallest 2-norm  $\|x\|$  is returned.

Parameters:  $a : (M, N)$  *array\_like*

“Coefficient” matrix.

$b : \{(M,), (M, K)\}$  *array\_like*

Ordinate or “dependent variable” values. If  $b$  is two-dimensional, the least-squares solution is calculated for each of the  $K$  columns of  $b$ .

# P2: Affine Camera Model RMS

After finding the calibrated camera matrix, you will compute the RMS error between the given  $N$  image corner coordinates and  $N$  corresponding calculated corner locations in `rms_error()`. Recall that

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

# Problem Outline

- Q1: Projective Geometry
- Q2: Affine Camera Calibration
- Q3: Single View Metrology



# Problem Outline

## Single View Geometry (45 points)

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) In Figure 2, we have identified a set of pixels to compute vanishing points in each image. Please complete `compute_vanishing_point()`, which takes in these two pairs of points on parallel lines to find the vanishing point. You can assume that the camera has zero skew and square pixels, with no distortion. **[5 points]**
- (b) Using three vanishing points, we can compute the intrinsic camera matrix used to take the image. Do so in `compute_K_from_vanishing_points()`. **[10 points]**
- (c) Is it possible to compute the camera intrinsic matrix for any set of vanishing points? Similarly, is three vanishing points the minimum required to compute the intrinsic camera matrix? Justify your answer. **[5 points]**

# Vanishing Points



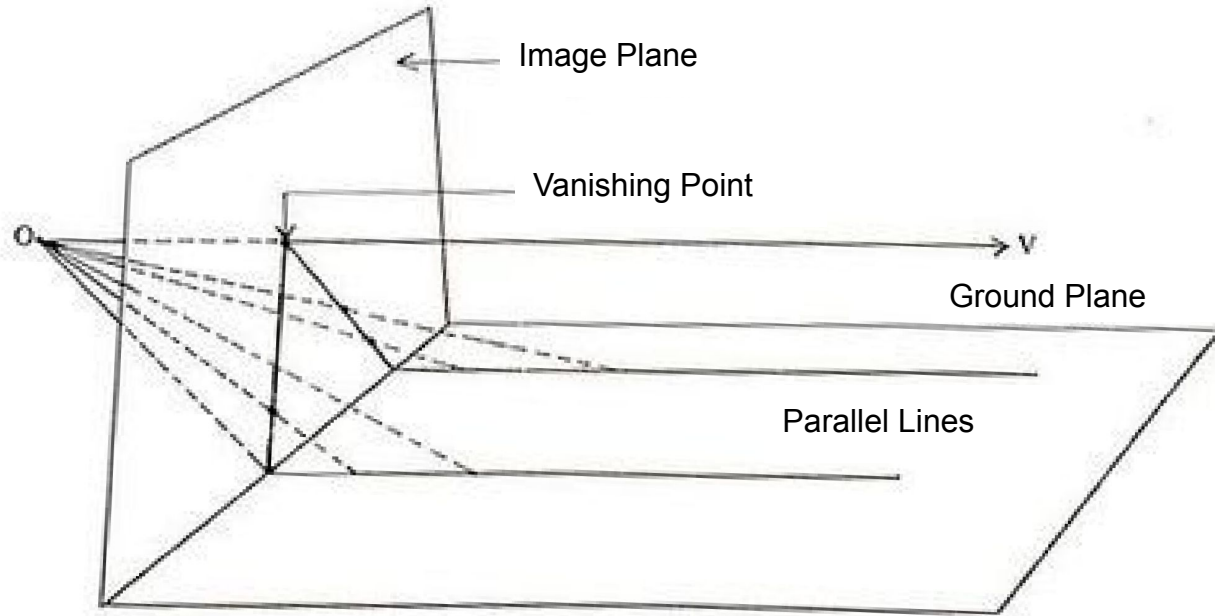


# Vanishing Points

- In 3D space, points at infinity are defined as the intersection of parallel lines, which have direction  $d$
- In the image plane, parallel lines meet at the vanishing point  $v$ .
- With camera intrinsic matrix as  $K$ , we have

$$v = Kd$$

# Vanishing Points



# Vanishing Points to Compute K

- Course notes “Single View Metrology”
- HZ (2nd edition) Page 223-226

## Course Notes

### Course Notes

In addition to the slides on the geometry-related topics of the first few lectures, we are also providing a self-contained notes for this course, in which we will go into greater detail about material covered by the course. Hopefully, this makes the content both more accessible and digestible by a wider audience.

[Course Notes 1: Camera Models](#)

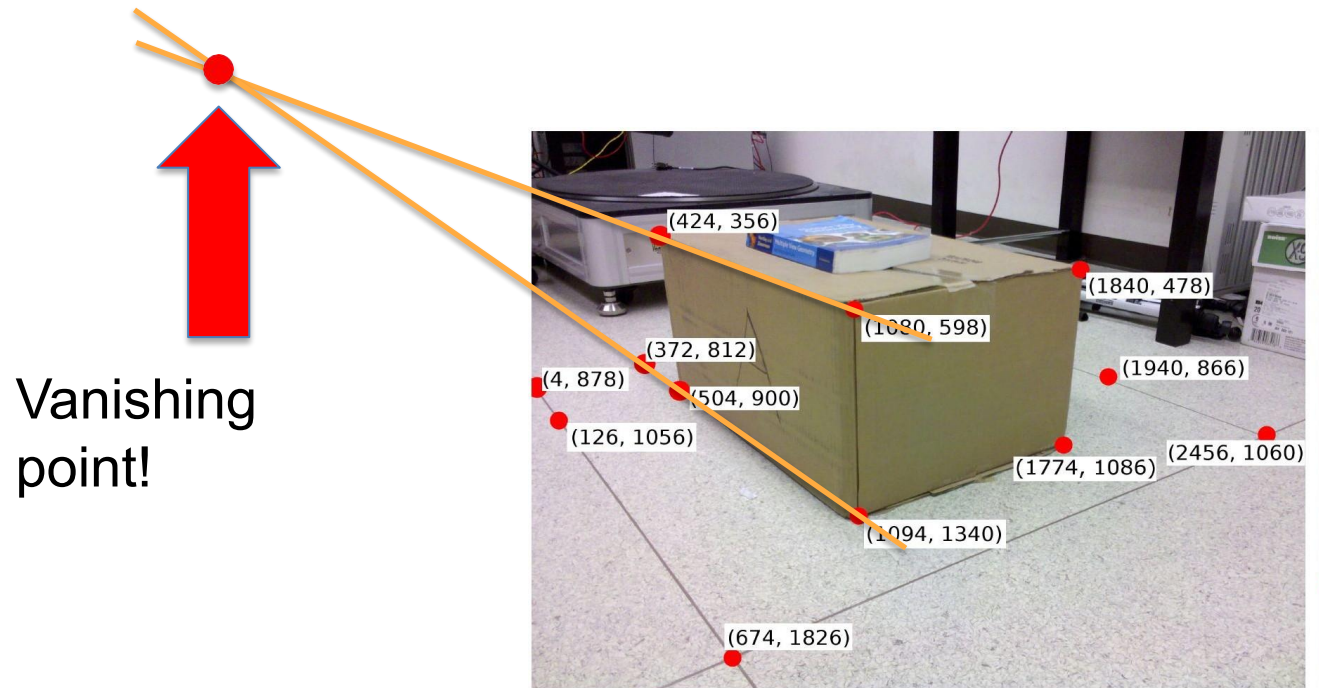
[Course Notes 2: Single View Metrology](#)

[Course Notes 3: Epipolar Geometry](#)

[Course Notes 4: Stereo Systems](#)

[Course Notes 5: Active and Volumetric Stereo](#)

# Calculating Vanishing Point



(a) Image 1 (1.jpg) with marked pixels

# Calculating Vanishing Point

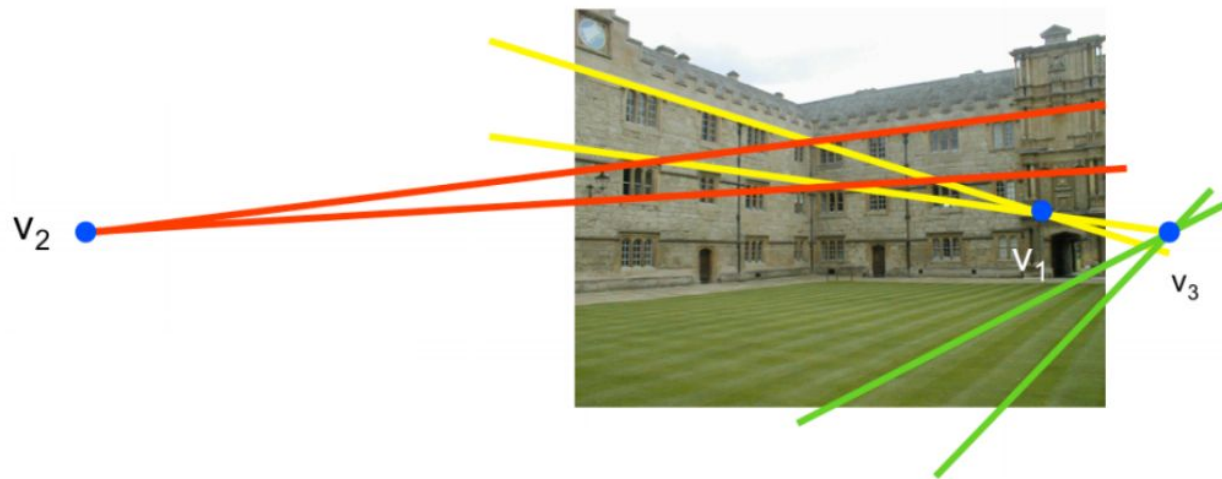


Figure 6: The example setup with three vanishing points for a set of mutually perpendicular planes.

# Calculating Vanishing Point

- Points in  $L_1$ :  $(x_1, y_1), (x_2, y_2) \rightarrow m_1 = (y_2 - y_1)/(x_2 - x_1)$
- Points in  $L_2$ :  $(x_3, y_3), (x_4, y_4) \rightarrow m_2 = (y_4 - y_3)/(x_4 - x_3)$
- Intersection of  $L_1$  and  $L_2$ : Vanishing Point

# Calculating Vanishing Point

```
# Part A: Compute vanishing points
```

```
v1 = compute_vanishing_point(np.array([[674,1826],[2456,1060],[1094,1340],[1774,1086]]))
```

```
v2 = compute_vanishing_point(np.array([[674,1826],[126,1056],[2456,1060],[1940,866]]))
```

```
v3 = compute_vanishing_point(np.array([[1094,1340],[1080,598],[1774,1086],[1840,478]]))
```

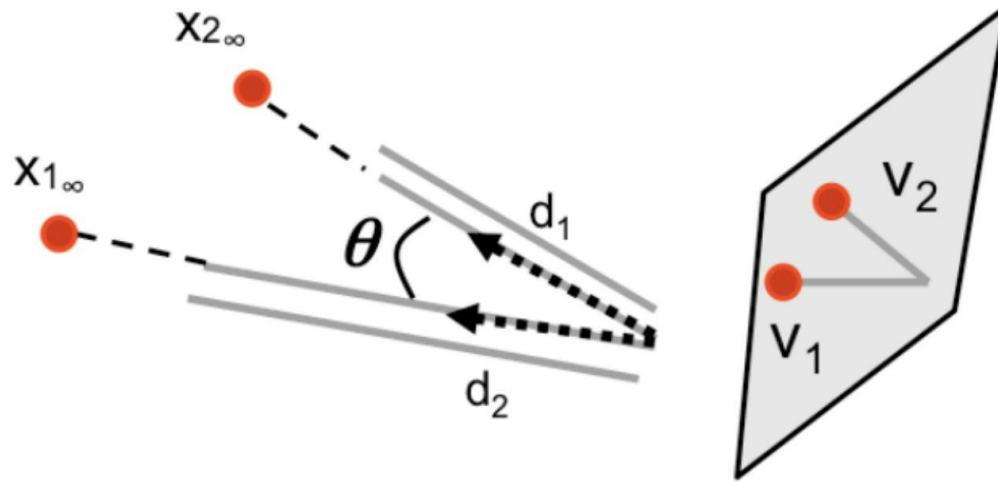
```
v1b = compute_vanishing_point(np.array([[314,1912],[2060,1040],[750,1378],[1438,1094]]))
```

```
v2b = compute_vanishing_point(np.array([[314,1912],[36,1578],[2060,1040],[1598,882]]))
```

```
v3b = compute_vanishing_point(np.array([[750,1378],[714,614],[1438,1094],[1474,494]]))
```

```
# Part B: Compute the error metric
```

# Vanishing Points to Compute K





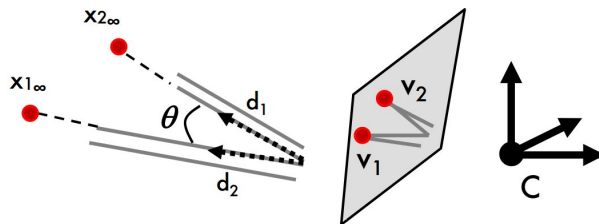
# Vanishing Points to Compute $K$

We can derive a useful relationship between parallel lines in 3D, their corresponding vanishing point in the image, and the camera parameters  $K, R, T$ . Let us define  $d = (a, b, c)$  as the direction of a set of 3D parallel lines in the camera reference system. These lines intersect to a point at infinity and the projection of such a point in the image returns the vanishing point  $v$ , which is defined by

$$v = Kd \tag{8}$$

# Vanishing Points to Compute K

- Define vanishing line  $d_1$  and  $d_2$ , vanishing points  $v_1$  and  $v_2$ . We have:



$$\cos \theta = \frac{v_1^T \omega v_2}{\sqrt{v_1^T \omega v_1} \sqrt{v_2^T \omega v_2}}$$

[Eq. 28]

$$\omega = (K K^T)^{-1}$$

[Eq. 30]

If  $\theta = 90$

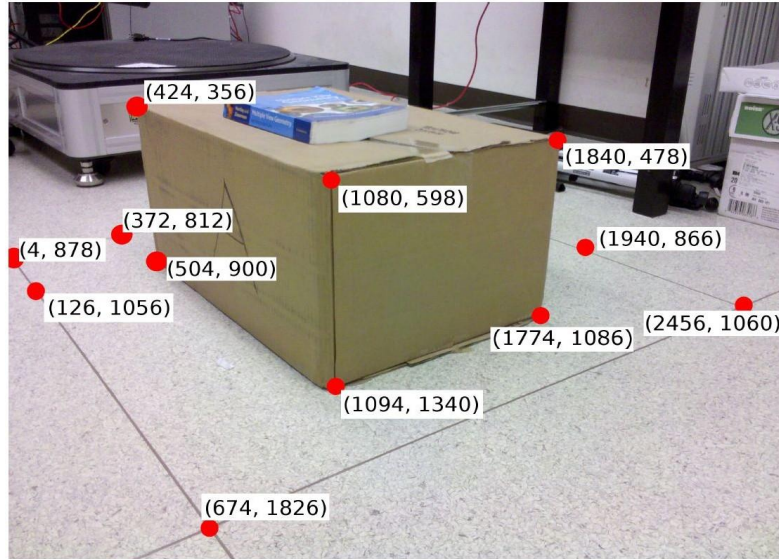
$\rightarrow$

$$v_1^T \omega v_2 = 0$$

[Eq. 29]

Scalar equation

# Vanishing Points to Compute K



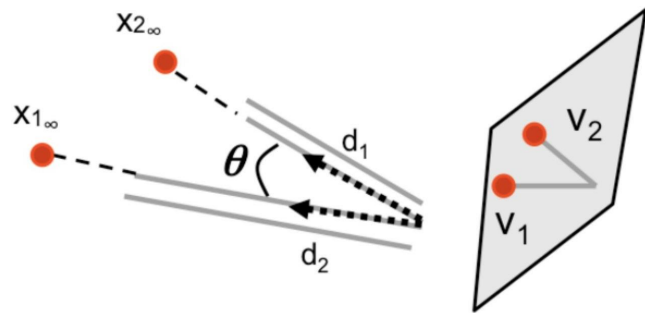
(a) Image 1 (1.jpg) with marked pixels

# Vanishing Points to Compute K

- Define vanishing line  $d_1$  and  $d_2$ , vanishing points  $v_1$  and  $v_2$ . We have:  $d = \frac{K^{-1}v}{\|K^{-1}v\|}$

$$\begin{aligned}\cos \theta &= \frac{d_1 \cdot d_2}{\|d_1\| \|d_2\|} \\ &= \frac{v_1^T \omega v_2}{\sqrt{v_1^T \omega v_1} \sqrt{v_2^T \omega v_2}}\end{aligned}$$

, where  $\omega = (K K^T)^{-1}$



If  $d_1$  and  $d_2$  are orthogonal,  $v_1^T \omega v_2 = 0$

# Vanishing Points to Compute K

- $\omega = (K K^T)^{-1}$ 
  - Matrix  $\omega$  is the projective transformation in the image plane of an absolute conic in 3D

$$\omega = \begin{bmatrix} \omega_1 & \omega_2 & \omega_4 \\ \omega_2 & \omega_3 & \omega_5 \\ \omega_4 & \omega_5 & \omega_6 \end{bmatrix}$$

# Vanishing Points to Compute K

- We assume the camera has zero skew and square pixels
  - Zero skew:  $\omega_2 = 0$
  - Square pixels:  $\omega_1 = \omega_3$

$$\omega = (K K^T)^{-1}$$

$$\omega = \begin{bmatrix} \omega_1 & 0 & \omega_4 \\ 0 & \omega_1 & \omega_5 \\ \omega_4 & \omega_5 & \omega_6 \end{bmatrix}$$

Camera matrix K

$$P' = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

# Compute Angle Between Planes

- Vanishing lines  $L_1$  and  $L_2$
- $L_1 = v_1 \times v_2$ ;  $L_2 = v_3 \times v_4$ 
  - $v_1$  and  $v_2$  = vanishing points corresponding to one plane
  - $v_3$  and  $v_4$  for the other plane

$$\cos\theta = \frac{l_1^T \omega^* l_2}{\sqrt{l_1^T \omega^* l_1} \sqrt{l_2^T \omega^* l_2}}$$

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

# Rotation Matrix using Vanishing Points

- Find corresponding vanishing points from both images ( $v_1, v_2, v_3$ ) and ( $v_1', v_2', v_3'$ )
- Calculate directions of vanishing points:

$$- v = K d \rightarrow \mathbf{d} = \frac{K^{-1} \mathbf{v}}{\|K^{-1} \mathbf{v}\|}$$

- $d_i' = R d_i$ , where
  - $d_i'$  = direction of the  $i^{\text{th}}$  vanishing point in second image
  - $d_i$  = direction of the  $i^{\text{th}}$  vanishing point in first image



Thank You

Questions?

