Computer Vision

---*Camera Calibration II*

Dr. WU Xiao Jun

2019.9.18

Camera Parameters

- A projection matrix can be written explicitly as a function of its five intrinsic parameters $(\alpha, \beta, u_0, v_0, \theta)$ and its six extrinsic ones (three angles defining $\boldsymbol{R}$) and three components of translation vector $\boldsymbol{t}$.

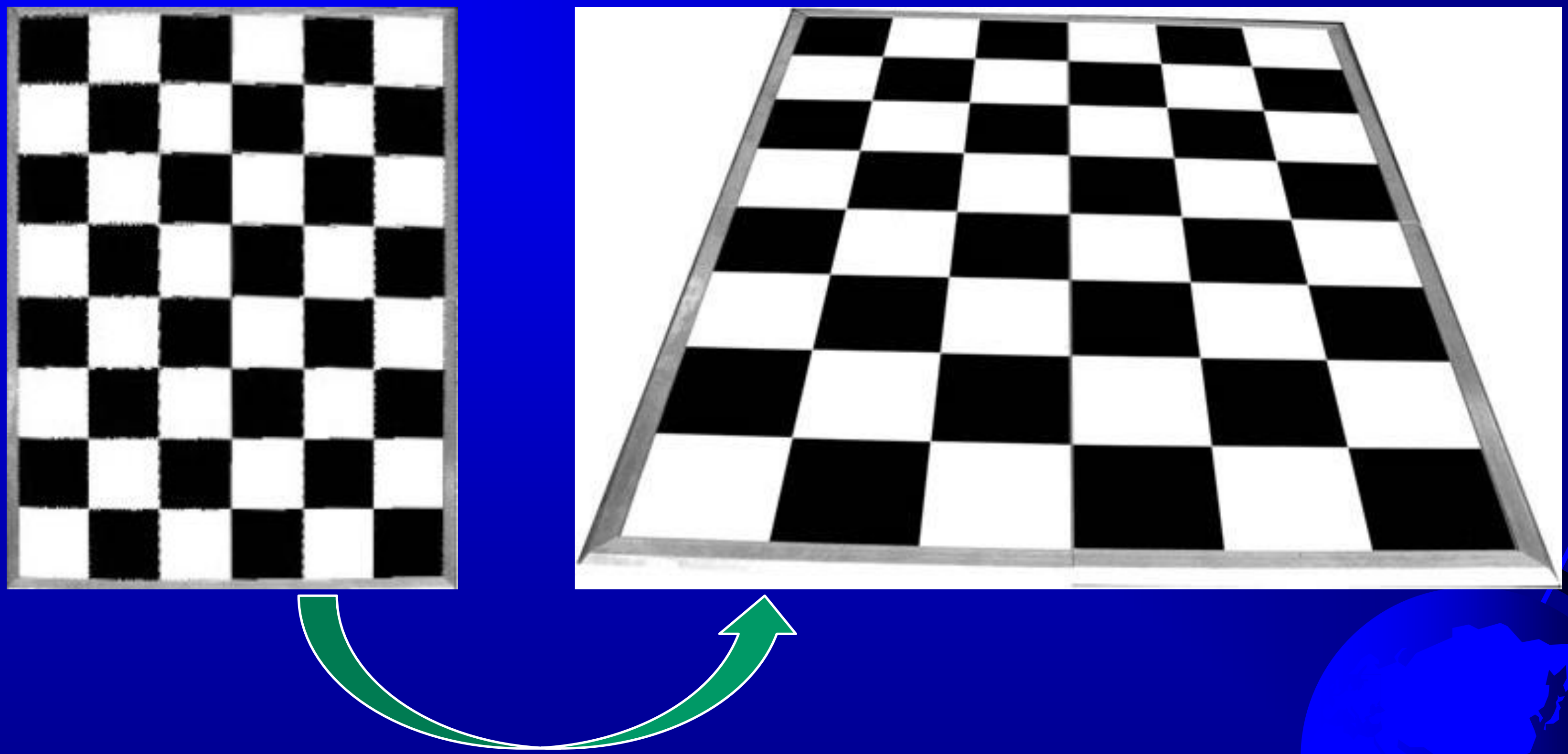
$$\mathcal{M} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix}$$

$$\mathcal{M} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T + u_0 \boldsymbol{r}_3^T & \alpha t_x + u_0 t_z \\ \beta \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \beta t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix}$$

$$\boldsymbol{p} = \frac{1}{z} \mathcal{M} \boldsymbol{P}$$

$$\begin{cases} u = \dfrac{\boldsymbol{m}_1 \cdot \boldsymbol{P}}{\boldsymbol{m}_3 \cdot \boldsymbol{P}} \\ u = \dfrac{\boldsymbol{m}_1 \cdot \boldsymbol{P}}{\boldsymbol{m}_3 \cdot \boldsymbol{P}} \end{cases}$$

Homography(单应)



Projective mapping

Homography(单应)

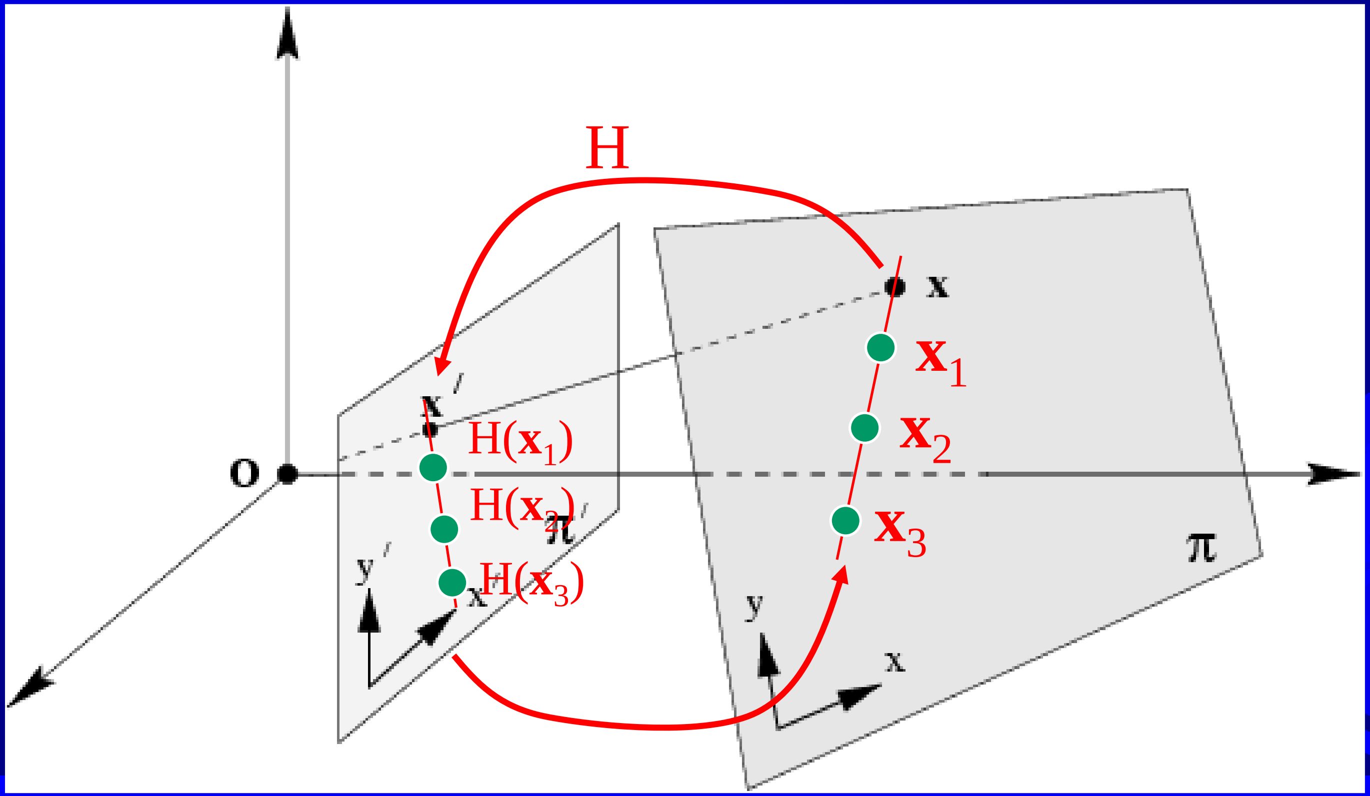
- General definition
  - A homography is an non-singular, line preserving, projective mapping $H: P^n \rightarrow P^n$
    - It is represented by a square (n+1)—dimension matrix with $(n+1)^2-1$ DoF
    - Note: homographies are not restricted to P^2
    - Homography=projective transformation=projectivity=collineation

Homography(单应)

- 2D homography
  - Definition:

Line preserving

A 2D *homography* is an invertible mapping h from P^2 to itself such that three points x_1, x_2, x_3 lie on the same line if and only if $h(x_1), h(x_2), h(x_3)$ do.



Homography(单应)

- 2D homography

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} fr_{11} & fr_{12} & ft_x \\ fr_{21} & fr_{22} & ft_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

Homography H (planar projective transformation)

Homography(单应)

- 2D homography
  - Theorem:

A mapping h: $P^2 \rightarrow P^2$ is a homography if and only if there exist a non-singular 3x3 matrix **H** such that for any point in P^2 represented by a vector **x** it is true that $h(x)=\mathbf{H}x$

  - Definition: homography

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad \text{or} \quad \mathrm{x}' = \mathbf{H}\,\mathrm{x}$$

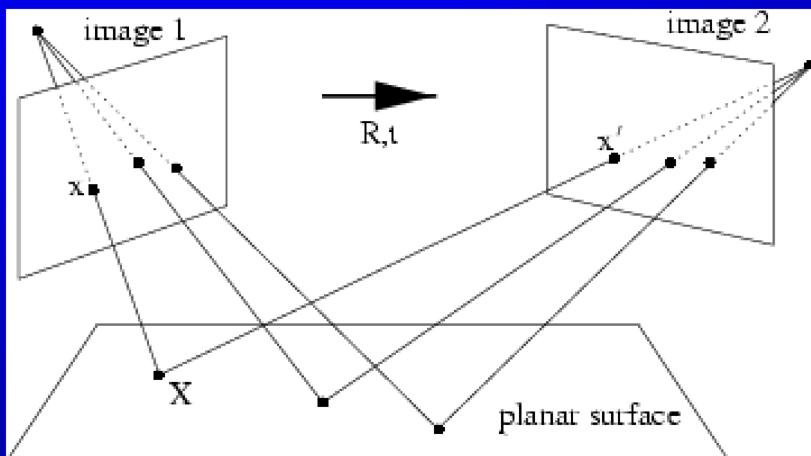
8DOF

Homography(单应)

- 2D homography
  - Homographies in computer vision
  - Rotating/translating camera, planar world

H

H



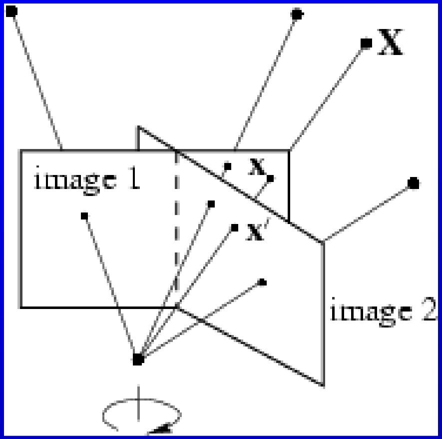
H

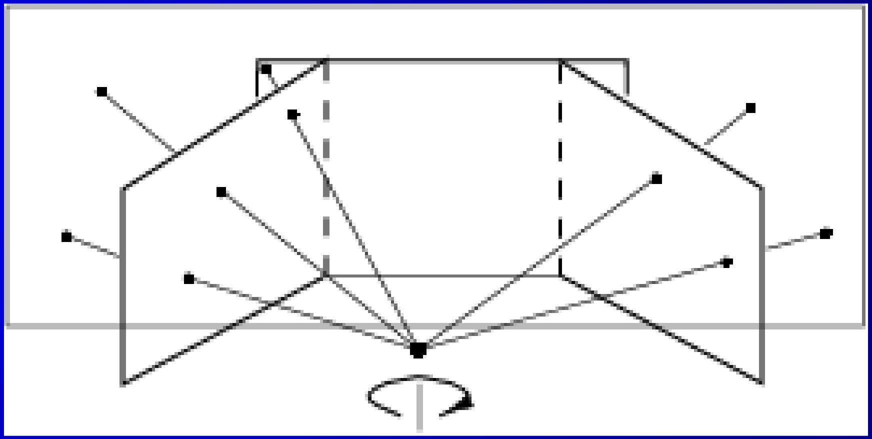
What happens to the P-matrix, if Z is assumed zero?

$$(x, y, 1)^T = x \propto PX = K[r_1 r_2 \cancel{r_3} t]\begin{pmatrix} X \\ Y \\ \cancel{0} \\ 1 \end{pmatrix} = H\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

Homography(单应)

- 2D Homography
  - Rotating camera, arbitrary world





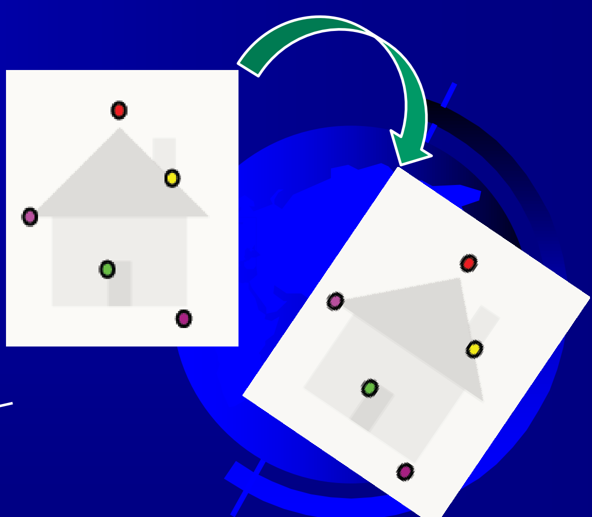
Homography(单应)

- 2D homography
  - Homography Transformation Hierarchy
  - Transformation hierarchy: isometries(iso=same, metric=measure)

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \varepsilon\cos\theta & -\sin\theta & t_x \\ \varepsilon\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad \varepsilon = \pm 1$$

orientation preserving: $\varepsilon = 1$
orientation reversing: $\varepsilon = -1$

$$\mathrm{x}' = \mathbf{H}_E\,\mathrm{x} = \begin{bmatrix} \mathbf{R} & \mathrm{t} \\ 0^{\mathsf{T}} & 1 \end{bmatrix} \mathrm{x} \qquad \mathbf{R}^{\mathsf{T}}\mathbf{R} = \mathbf{I}$$



3DOF (1 rotation, 2 translation)
Special cases: pure rotation, pure translation

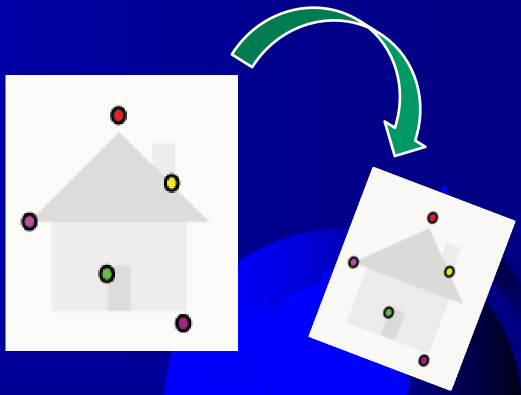
Invariants: length, angle, area

Homography(单应)

- 2D homography
  - Homography Transformation Hierarchy
  - Transformation hierarchy: scaling

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

4DOF (1 rotation, 2 translation, 1 scale)
Special cases: pure rotation, pure translation



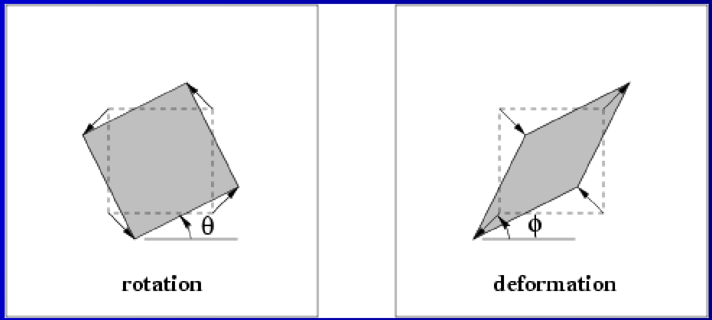
Invariants: angle

Homography(单应)

- 2D homography
  - Homography Transformation Hierarchy
  - Transformation hierarchy: affinities

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\mathrm{x}' = \mathbf{H}_A \, \mathrm{x} = \begin{bmatrix} \mathbf{A} & \mathrm{t} \\ 0^\mathsf{T} & 1 \end{bmatrix} \mathrm{x}$$



$$\mathbf{A} = \mathbf{R}(\theta)\mathbf{R}(-\phi)\mathbf{D}\mathbf{R}(\phi) \qquad \mathbf{D} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

6DOF (2 scale, 2 rotation, 2 translation)
Non-isotropic scaling! (2DOF: scale ratio and orientation)

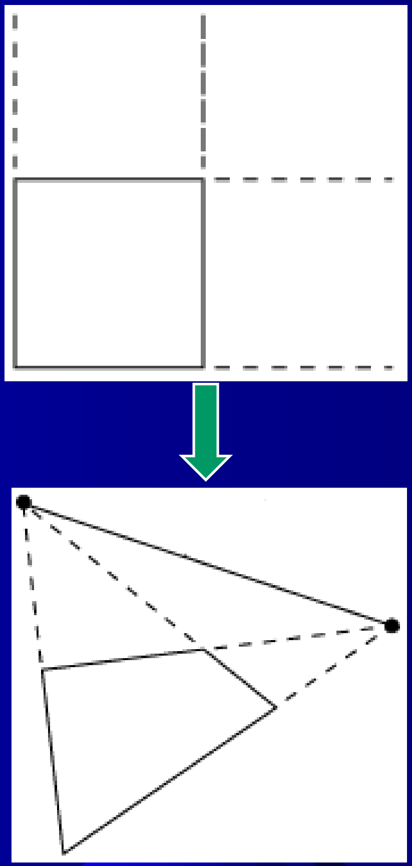
Invariants: parallel lines, ratios of parallel lengths, ratios of areas

Homography(单应)

- 2D homography
  - Homography Transformation Hierarchy
  - Transformation hierarchy: homographies

$$\mathbf{H}_P = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} = \begin{pmatrix} \mathbf{A} & \vec{t} \\ \vec{v}^T & v \end{pmatrix}$$

$$\mathrm{x}' = \mathbf{H}_P \, \mathrm{x} = \begin{bmatrix} \mathbf{A} & \mathrm{t} \\ \mathrm{v}^{\mathsf{T}} & v \end{bmatrix} \mathrm{x} \qquad \mathrm{v} = (v_1, v_2)^{\mathsf{T}}$$



8DOF (2 scale, 2 rotation, 2 translation, 2 line at infinity)

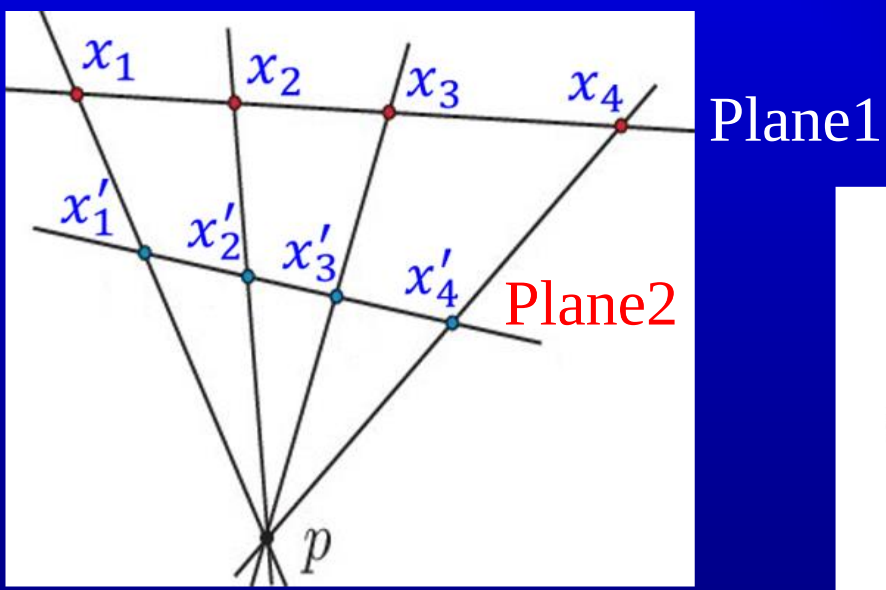
Invariants: cross-ratio of four points on a line (ratio of ratio)

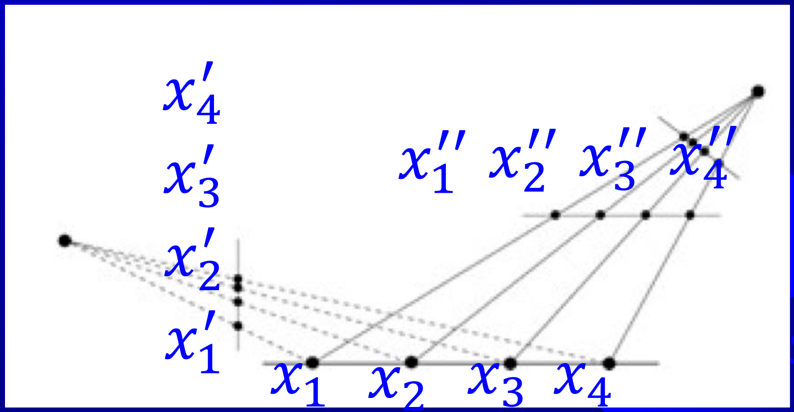
Allows to observe vanishing points, horizon

Homography(单应)

- 2D homography
  - Cross ratio (交比)

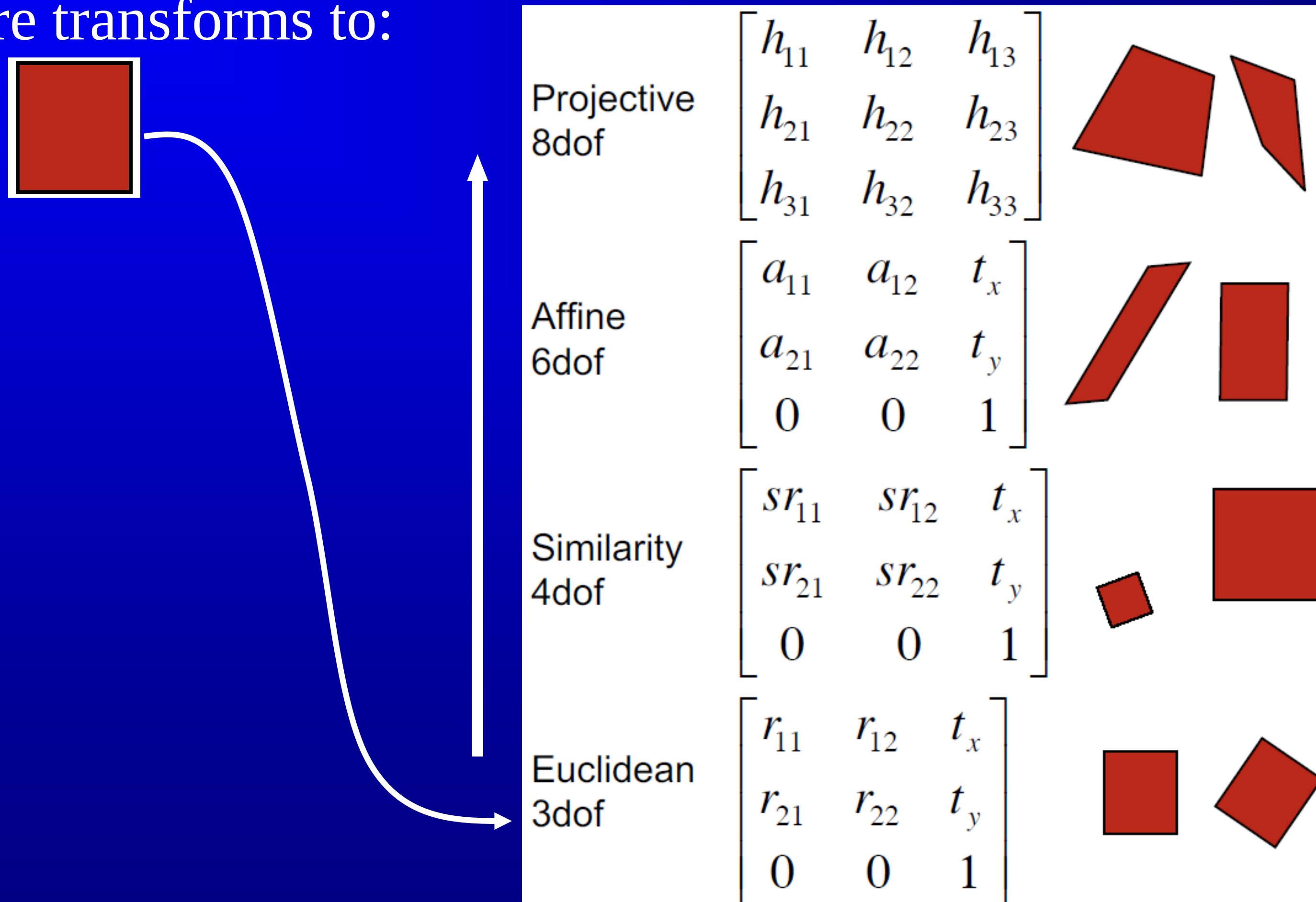
$$Cross(x_1,x_2;x_3,x_4)=\frac{(x_3-x_1)(x_4-x_2)}{(x_3-x_2)(x_4-x_1)}=\frac{(x_3'-x_1')(x_4'-x_2')}{(x_3'-x_2)(x_4'-x_1')}$$





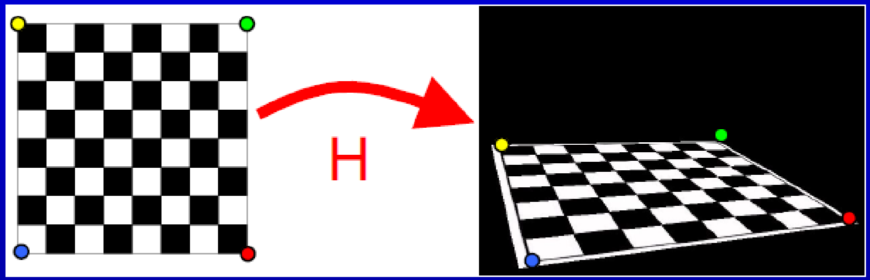
Homography(单应)

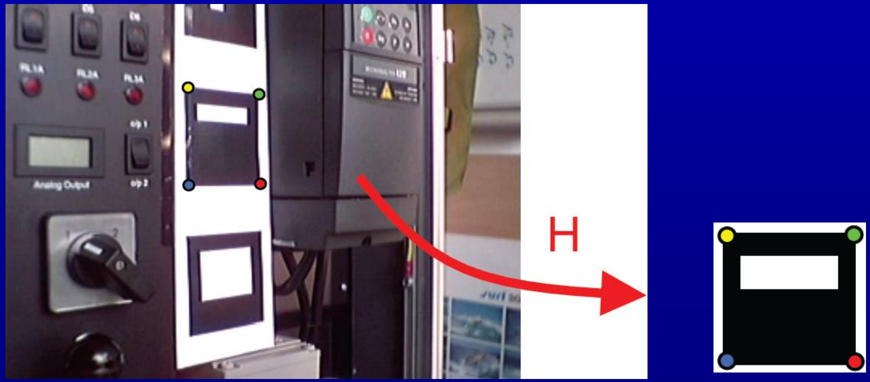
- 2D homography
  - Homography Transformation Hierarchy
  - A square transforms to:

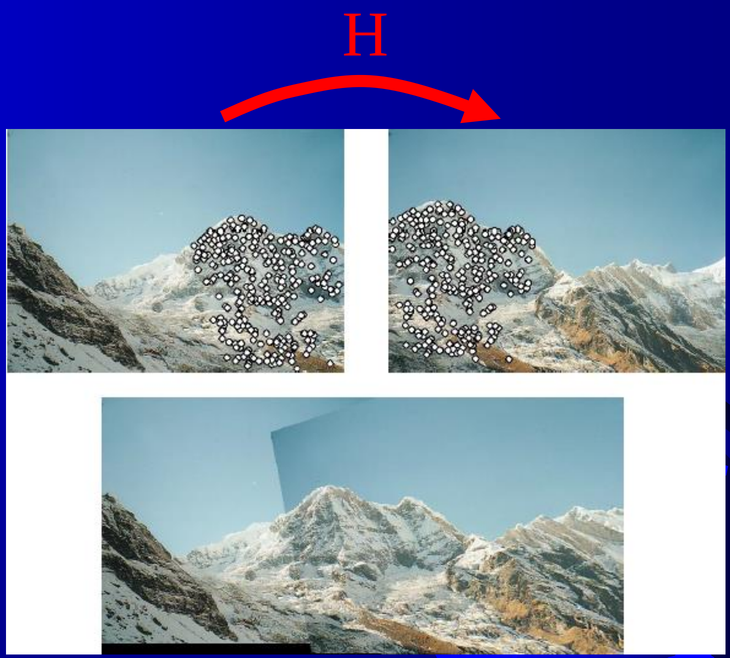


Homography(单应)

- 2D homography
  - How to estimate a homography from point correspondences?
  - Estimate homography from point correspondences between:
    - Two images
    - Model plane and image
  - Assumption: planar motion.







Homography(单应)

- 2D homography
  - To estimate H, we start from the equation $x' = Hx$. In homogeneous coordinates we the following constraint:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

  - Homography estimation in 2D plane, we set $z' = 1, z = 1$. We get

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homography(单应)

- 2D homography
  - Then, we can get

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}$$
$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

  - DoF of 2D homography is 8, we can set $h_{33} = 1$ or give a constraint to H,

$$h_{11}^2 + h_{12}^2 + h_{13}^2 + h_{21}^2 + h_{22}^2 + h_{23}^2 + h_{31}^2 + h_{32}^2 + h_{33}^2 = 1$$

Homography(单应)

- 2D homography
  - Setting $h_{33} = 1$

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}$$

  - Multiplying through by denominator

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

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

  - Rearrange

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

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

Homography(单应)

- 2D homography

$$
\begin{array}{rcccc}
 & \textbf{2N x 8} & \textbf{8 x 1} & & \textbf{2N x 1} \\
\begin{array}{r} \textbf{Point 1} \\ \\ \textbf{Point 2} \\ \\ \textbf{Point 3} \\ \\ \textbf{Point 4} \\ \\ \end{array}
&
\begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\
0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\
x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\
0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\
x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\
0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\
x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\
0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4'
\end{bmatrix}
&
\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}
& = &
\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} \\
\begin{array}{l} \textbf{additional} \\ \textbf{points} \end{array} & \vdots & & & \vdots
\end{array}
$$

Homography(单应)

- 2D homography

Linear equations

$$\underset{2N\times 8}{A}\ \underset{8\times 1}{h} = \underset{2N\times 1}{b}$$

Solve:

$$\underset{8\times 2N}{A^T}\ \underset{2N\times 8}{A}\ \underset{8\times 1}{h} = \underset{8\times 2N}{A^T}\ \underset{2N\times 1}{b}$$

$$\overbrace{(A^T\ A)}^{8\times 8}\ \underset{8\times 1}{h} = \overbrace{(A^T\ b)}^{8\times 1}$$

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

Matlab: $h = A \backslash b$

Homography(单应)

- 2D homography (Constraint ||**h**||=1)
  - From

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13}$$
$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23}$$

Get

$$h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' - h_{33}x' = 0$$
$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' - h_{33}y' = 0$$

Setting

$$\boldsymbol{h} = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^T$$
$$\boldsymbol{a}_x = (-x, -y, -1, 0, 0, 0, x'x, x'y, x')^T$$
$$\boldsymbol{a}_y = (0, 0, 0, -x, -y, -1, 0, 0, 0, y'x, y'y, y')^T$$

Get $\boldsymbol{a}_x^T\boldsymbol{h} = 0$

$\boldsymbol{a}_y^T\boldsymbol{h} = 0$

Homography(单应)

- 2D homography (Constraint ||**h**||=1)

$$\begin{array}{cccccc}
 & \textbf{2N x 9} & \textbf{9 x 1} & & \textbf{2N x 1} \\
\begin{array}{c}\textbf{4}\\ \textbf{P}\\ \textbf{O}\\ \textbf{I}\\ \textbf{N}\\ \textbf{T}\\ \textbf{S}\end{array} &
\begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' & -x_1' \\
0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' & -y_1' \\
 & & & & & & & & \\
 & & & & & & & & \\
 & & & & & & & & \\
 & & & & & & & &
\end{bmatrix} &
\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} & = &
\begin{bmatrix} 0 \\ 0 \\ \\ \\ \\ \\ \end{bmatrix} \\
\textbf{additional points} & \vdots & & & \vdots
\end{array}$$

Homography(单应)

- 2D homography (Constraint ||**h**||=1)

Homogeneous equations $\overset{\text{2Nx9}}{\mathbf{A}}\ \overset{\text{9x1}}{\mathbf{h}} = \overset{\text{2Nx1}}{\mathbf{0}}$

Solve: $\overset{\text{9x2N}}{\mathbf{A}^T}\ \overset{\text{2Nx9}}{\mathbf{A}}\ \overset{\text{9x1}}{\mathbf{h}} = \overset{\text{9x2N}}{\mathbf{A}^T}\ \overset{\text{2Nx1}}{\mathbf{0}}$

$$\overbrace{(\mathbf{A}^T\ \mathbf{A})}^{\text{9x9}}\ \overset{\text{9x1}}{\mathbf{h}} = \overset{\text{9x1}}{\mathbf{0}}$$

SVD of $\mathbf{A}^T\mathbf{A} = \mathbf{U}\ \mathbf{D}\ \mathbf{U}^T$

Let h be the column of U (unit eigenvector) associated with the smallest eigenvalue in D. (if only 4 points, that eigenvalue will be 0)

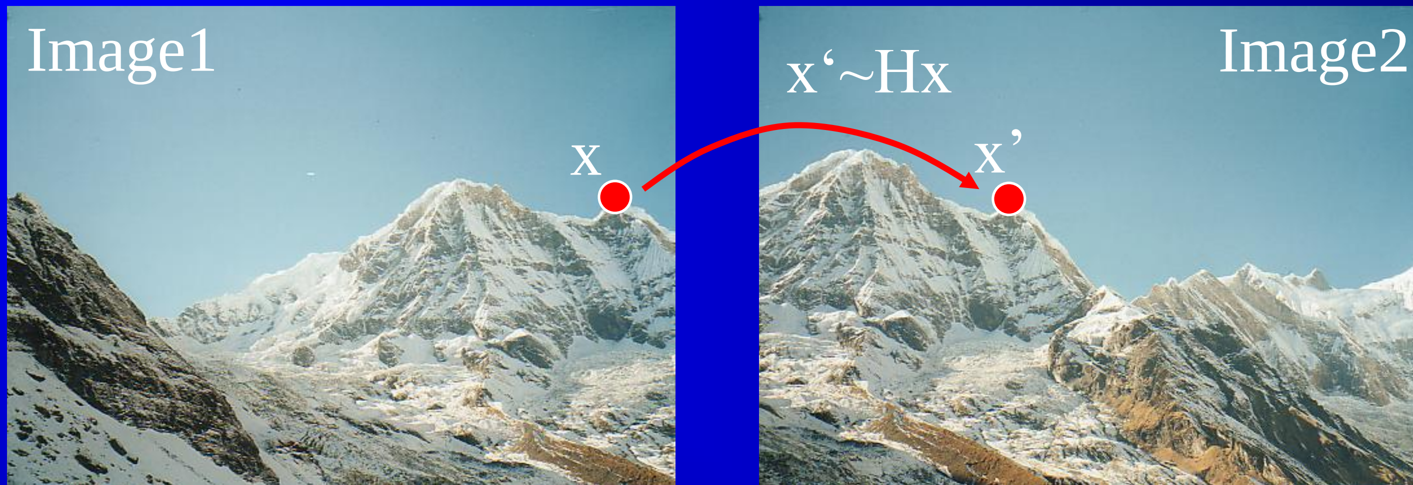
Homography(单应)

And what now?

What can we do when knowing the homography between two images?

Homography(单应)

- Application(1): panorama

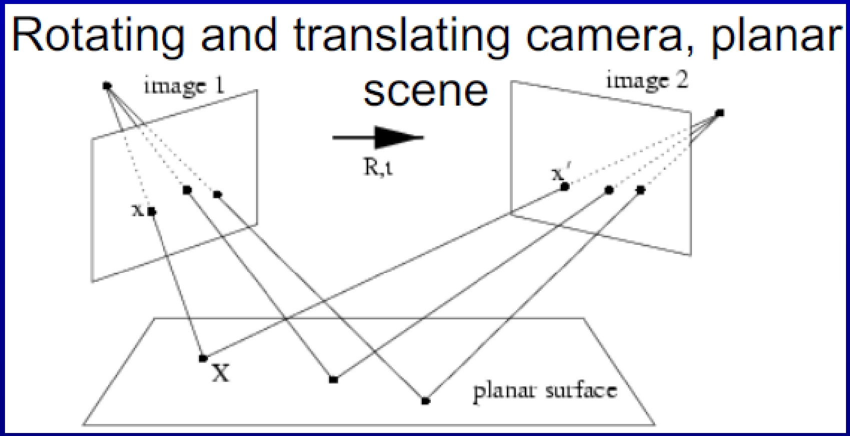


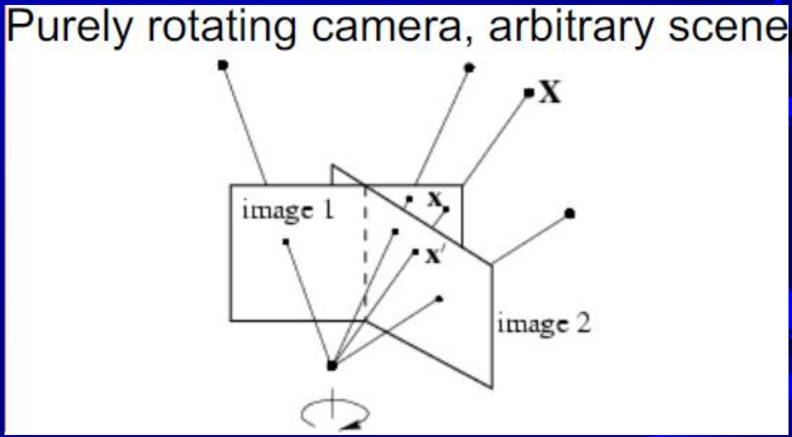
Panorama stitching:

1. Undistort images
2. Find point correspondences between images
3. Compute homography H
4. Resample:
   1. Loop over image 1
   2. Project into image 2 using H
   3. Bilinear interpolation in image 2

Homography(单应)

- Application(2): camera pose estimation
  - Assuming K (intrinsic calibration matrix) and H are known, derive the 3D camera pose (R and t)
  - Enables augmentation of 3D virtual objects (augmented reality)
    - Set virtual camera to real camera
    - Render virtual scene
    - Compose with real image
  - Enable localization/navigation
  - Recall the two cases of planar motion:





Homography(单应)

- Application(2): camera pose estimation
  - Enables augmentation of 3D virtual objects (augmented reality)
    - Set virtual camera to real camera
    - Render virtual scene
    - Compose with real image

Homography(单应)

- Application(2): camera pose estimation
  - Assuming all points lie in one plane with $Z = 0$: $\boldsymbol{X} = (X, Y, 0, 1)$

$$\boldsymbol{x} = \boldsymbol{PX} = K[\boldsymbol{r}_1 \ \ \boldsymbol{r}_2 \ \ 0 \ \ \boldsymbol{t}]\begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix}$$

$$= K[\boldsymbol{r}_1 \ \ \boldsymbol{r}_2 \ \ \boldsymbol{t}]\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

$$H = \lambda K[\boldsymbol{r}_1 \ \ \boldsymbol{r}_2 \ \ \boldsymbol{t}] \Longrightarrow K^{-1}H = \lambda[\boldsymbol{r}_1 \ \ \boldsymbol{r}_2 \ \ \boldsymbol{t}]$$

- $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are unit vectors $\Longrightarrow$ find λ.
- Use this to compute $\boldsymbol{t}$.
- Rotation matrices are orthogonal $\Longrightarrow$ find $\boldsymbol{r}_3$.

$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad (\boldsymbol{r}_1 \times \boldsymbol{r}_2) \quad \boldsymbol{t}]$$

Homography(单应)

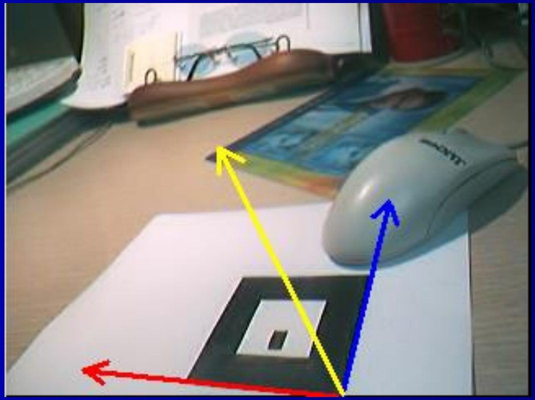
- Application(2): camera pose estimation
  - Problem
    - The vectors $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ might not yield the same λ.
  - Solution:
    - Use the average value.
  - Problem
    - The estimated rotation matrix might not be orthogonal.
  - Solution: orthogonalize R'
    - Objtain SVD $\Longrightarrow R' = UWV^T$
    - Set singular values to $W = 1 \Longrightarrow R' = UV^T$

Homography(单应)

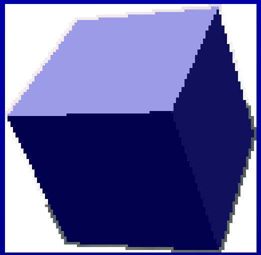
- Application(2): camera pose estimation
  - Marker tracker

Video-input

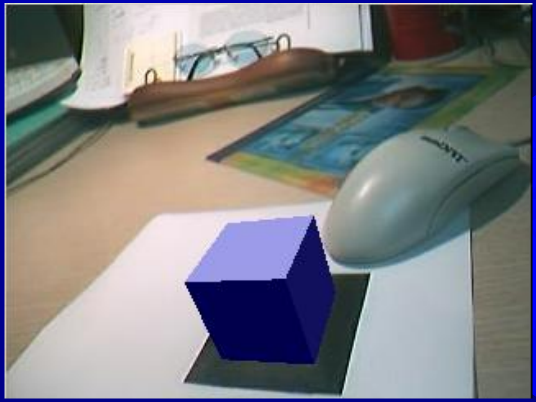
Pattern recognition (point correspondences from 4 corners



Homography → 3D pose



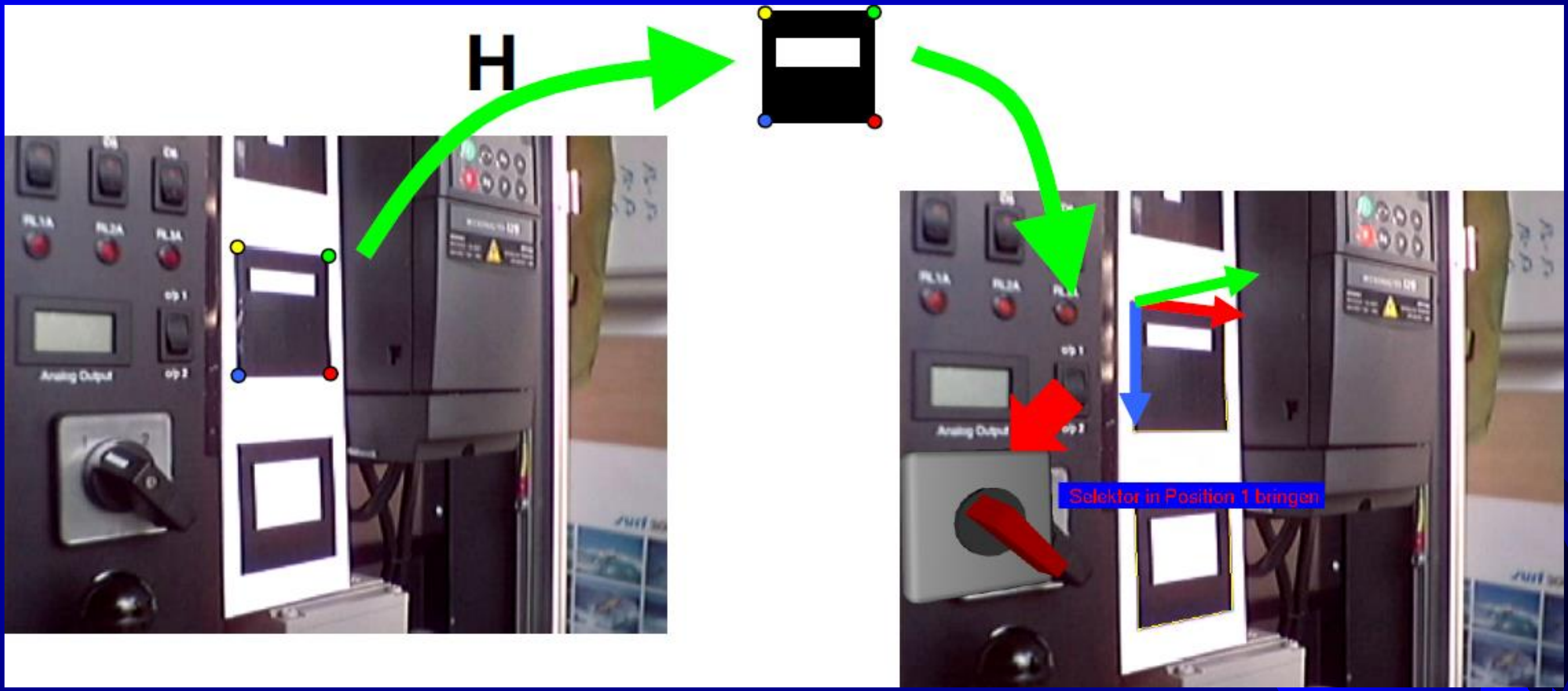
Rendering of the virtual object



Synthesis and overlay

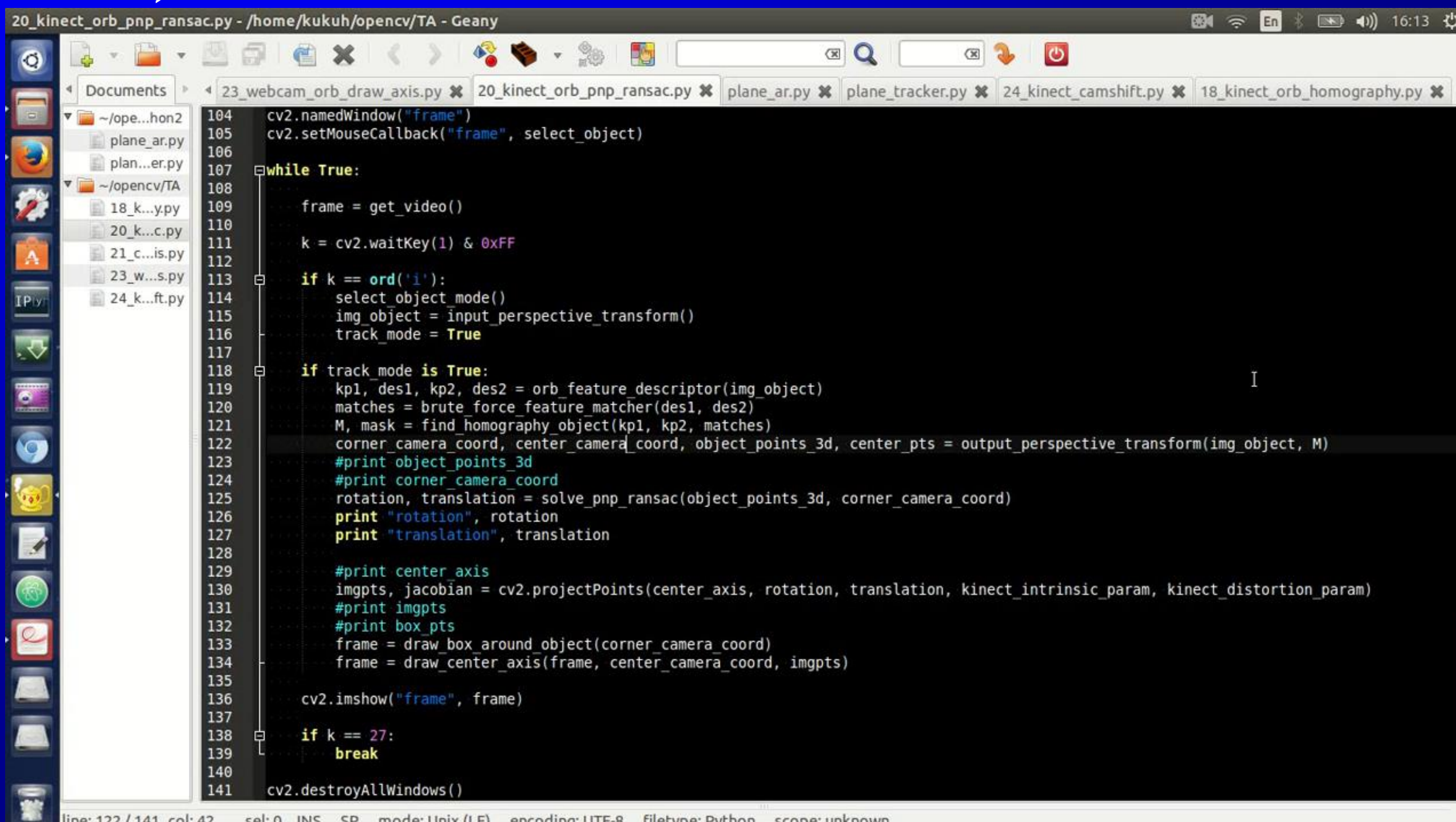
Homography(单应)

- Application(2): camera pose estimation
  - Planar scene (example marker tracker, applies to any planar scene):



Homography(单应)

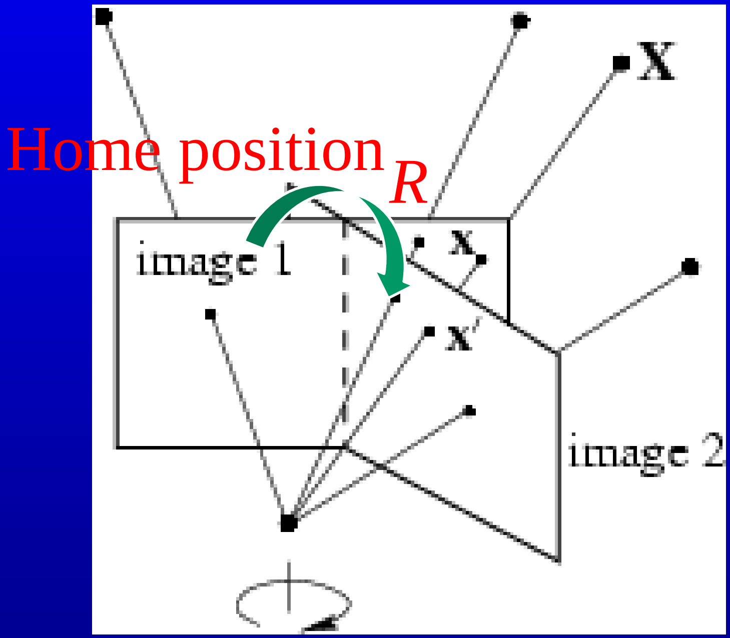
- Application(2): camera pose estimation
  - Planar scene (example marker tracker, applies to any planar scene):



```python
cv2.namedWindow("frame")
cv2.setMouseCallback("frame", select_object)

while True:

    frame = get_video()

    k = cv2.waitKey(1) & 0xFF

    if k == ord('i'):
        select_object_mode()
        img_object = input_perspective_transform()
        track_mode = True

    if track_mode is True:
        kp1, des1, kp2, des2 = orb_feature_descriptor(img_object)
        matches = brute_force_feature_matcher(des1, des2)
        M, mask = find_homography_object(kp1, kp2, matches)
        corner_camera_coord, center_camera_coord, object_points_3d, center_pts = output_perspective_transform(img_object, M)
        #print object_points_3d
        #print corner_camera_coord
        rotation, translation = solve_pnp_ransac(object_points_3d, corner_camera_coord)
        print "rotation", rotation
        print "translation", translation

        #print center_axis
        imgpts, jacobian = cv2.projectPoints(center_axis, rotation, translation, kinect_intrinsic_param, kinect_distortion_param)
        #print imgpts
        #print box_pts
        frame = draw_box_around_object(corner_camera_coord)
        frame = draw_center_axis(frame, center_camera_coord, imgpts)

    cv2.imshow("frame", frame)

    if k == 27:
        break

cv2.destroyAllWindows()
```

Homography(单应)

- 2D homography
  - Purely rotating camera



Home position $x = K[I \quad 0]X = KX$ (1)

Rotation by a matrix R

How to compute H?

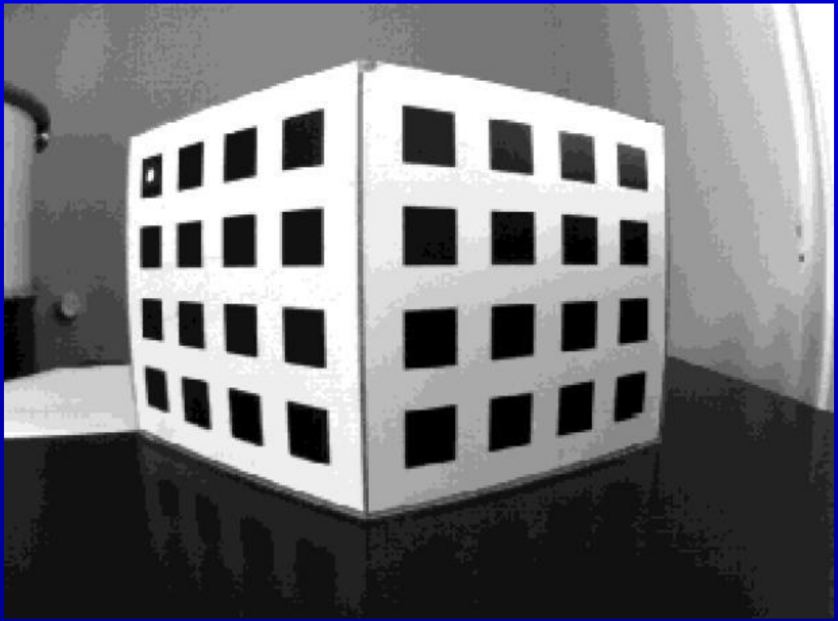
Homework1

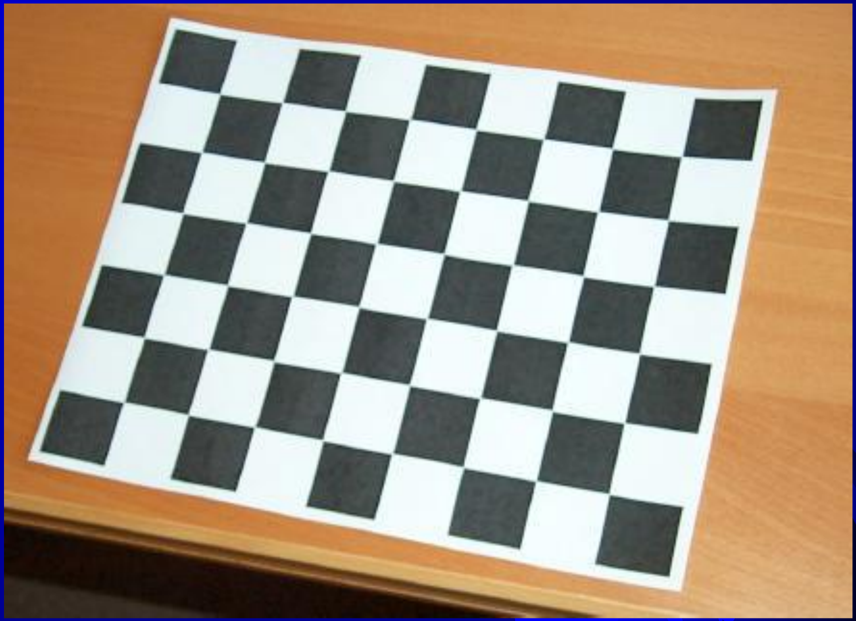
Camera Calibration

- Good calibration is important when we need reconstruct a world model.
- Interact with the world robot, hand-eye coordination
- Issues:
  - what is the camera matrix?(intrinsic+extrinsic)?
  - what are intrinsic and extrinsic parameters of the camera?
- General strategies
  - a set of features such as points or lines are known in some fixed world coordinate system.
  - view calibration object
  - identify image points
  - obtain camera matrix by minimizing error
  - obtain intrinsic parameters from camera matrix

Camera Calibration

- The problem: compute the camera intrinsic and extrinsic parameters using only observed camera data.



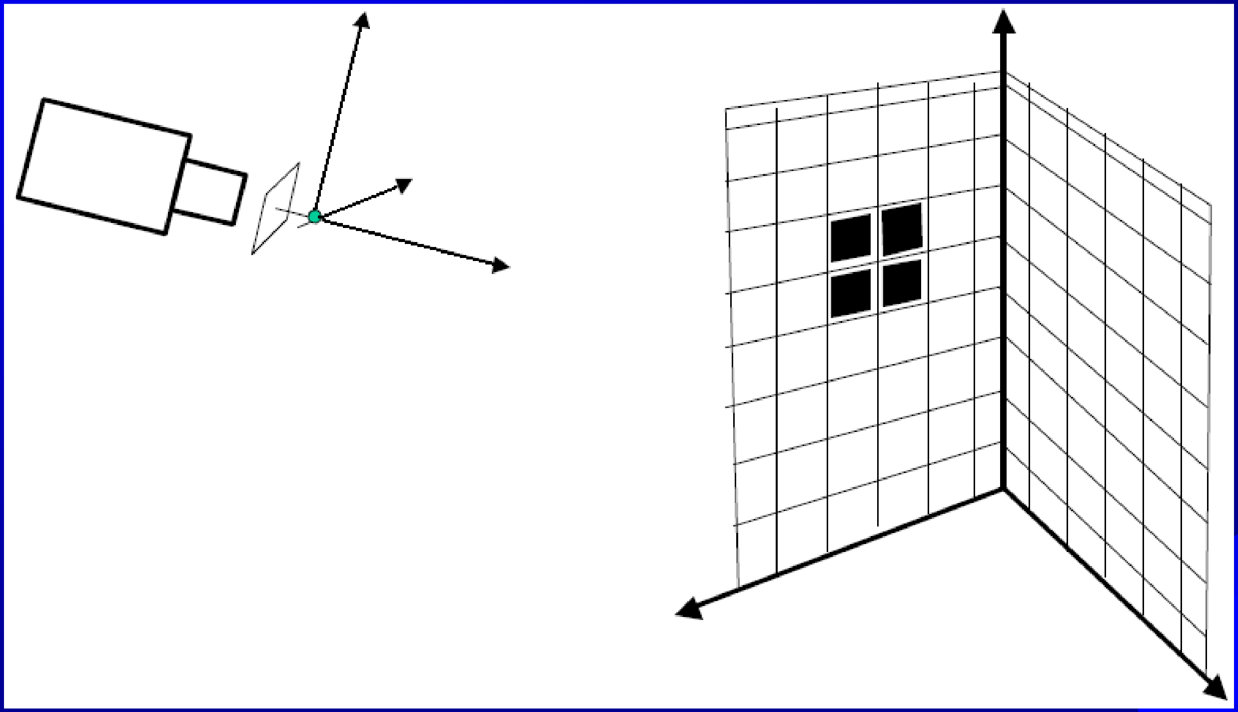


Calibration Classifications

- Calibration pattern based method
  - Feature: Utilize the structural information of the scene. The calibration target is often used.
  - Pros: Can be employed in any camera model with high calibration accuracy.
  - Cons: The calibration procedure is complex and the structural information should be highly accurate.
- Camera self-calibration method.
  - Feature: Using the correspondences between multi-images to calibrate.
  - Pro: Only setup the correspondences between multi-images with high flexibility and potential use in wide range of applications.
  - Con: Nonlinear, low robustness.

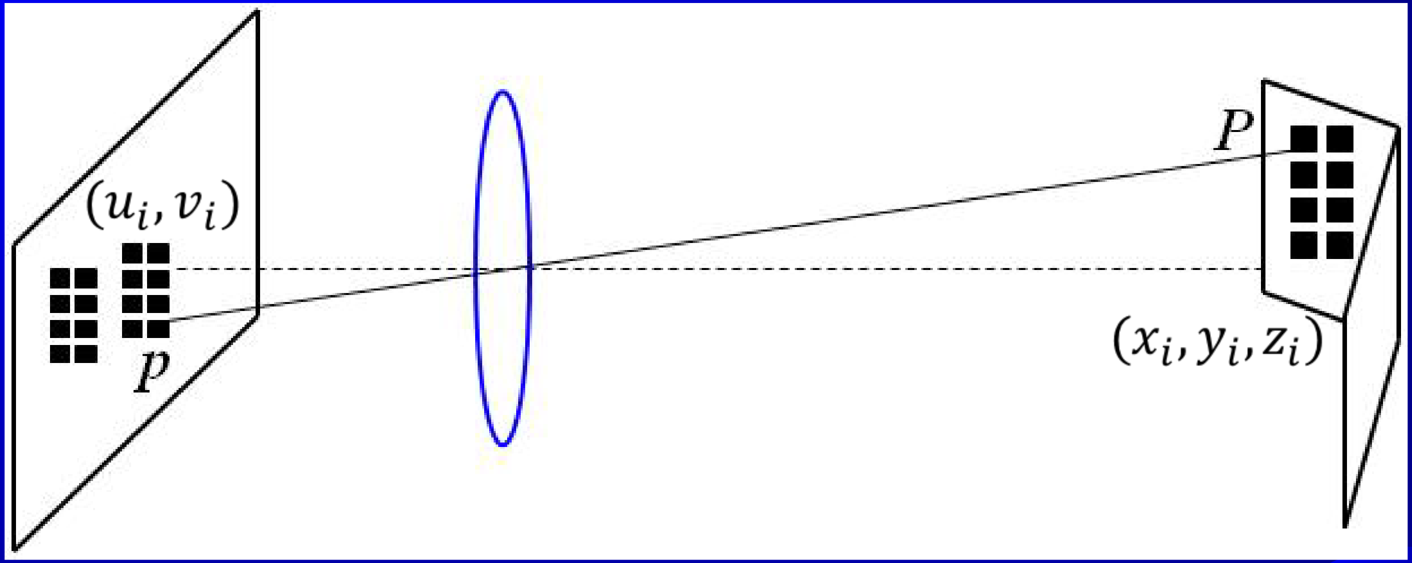
Linear camera calibration

- Assume we have known the image positions (u_i, v_i) of n points P_i (automatically or by hand)



Linear camera calibration

- Assume we have known the image positions (u_i, v_i) of n points P_i (automatically or by hand)



$$\boldsymbol{Ax} = 0 \quad \text{or} \quad \boldsymbol{Ax} = \boldsymbol{b}$$

Linear camera calibration

1. p linear equation in q unknows:

$$\left\{\begin{array}{c} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1q}x_q = y_1 \\ u_{21}x_1 + u_{22}x_2 + \cdots + u_{2q}x_q = y_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ u_{p1}x_1 + u_{p2}x_2 + \cdots + u_{pq}x_q = y_p \end{array}\right. \Leftrightarrow \mathcal{U}\boldsymbol{x} = \boldsymbol{y}$$

2. when $p < q$, the solution forms a $(p - q)$-dimensional vector subspace of $\mathbb{R}^q$
3. when $p = q$, there is a unique solution
4. when $p > q$, there is no solution

Linear camera calibration

- Linear least square
- Non-linear least square
- Newton's method: square system of nonlinear equation
- The Gaussian-Newton and Levenberg Marquardt algorithm

(2)

Linear camera calibration

- Levenberg Marquardt algorithm
  - First put forward by Kenneth Levernberg in 1994 to provide solutions for problems called as Nolinear linear squares minimization.
  - Update function was a blend of the characteristics of the Speedest descent and Newton’s method.
  - Imporved by Donald Marquardt in 1963 who incorporated the estimated local curvature information into the update function.
  - The original algorithm was put into the trust-region framework by More and Sorensen in 1983.
  - Used in Non-linear least square programming (NLP) with(2)
    - Unconstrained or
    - Unbounded constrained problems.

Linear camera calibration

- Levenberg Marquardt algorithm
  - http://people.duke.edu/~hpgavin/ce281/lm.pdf

$$[\mathcal{J}_f^T \mathcal{J}_f(\boldsymbol{x}) + \mu Id]\delta \boldsymbol{x} = -\mathcal{J}_f^T f(\boldsymbol{x})$$

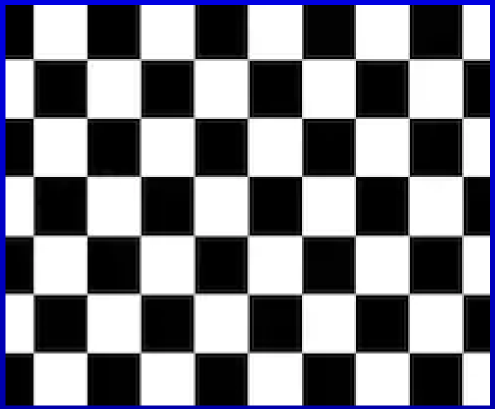
μ is allowed to vary at each iteration, we obtain the Levenberg-Marquardt algorithm.

It is more robust and can be used when the Jacobian $\mathcal{J}_f$ does not have maximal rank and its pseudoinverse does not exist.

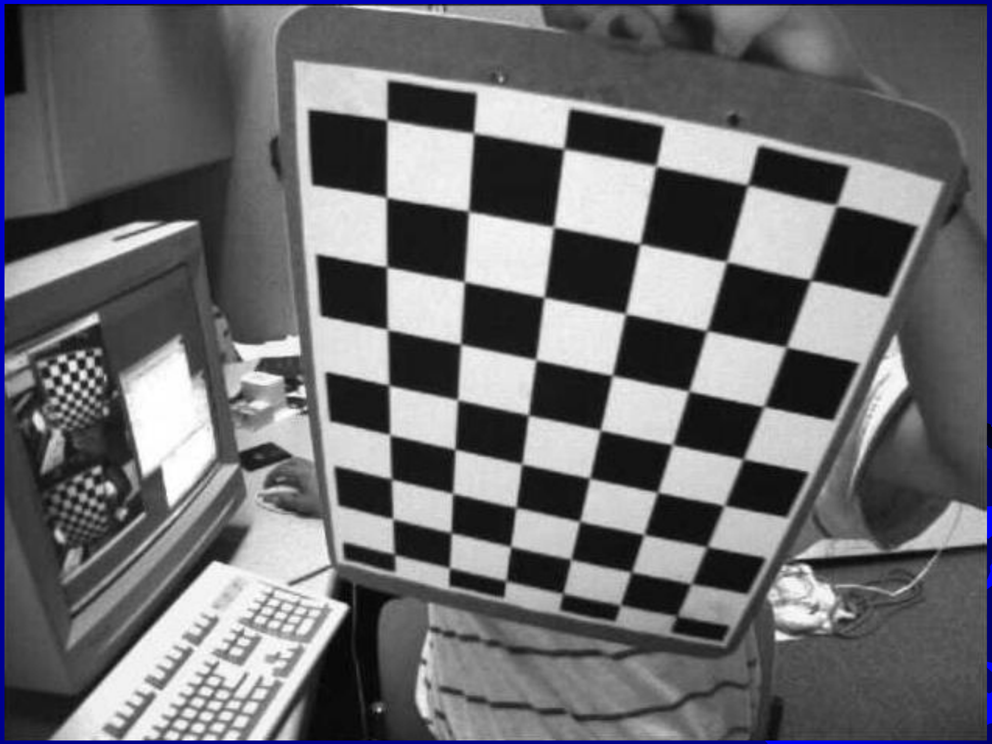
(2)

Homework2

- Using the introduced method to compute Homography matrix by using the following image.



Grid size 3cmx3cm



See You