6.2 Pose Estimation & Calibration

Outline

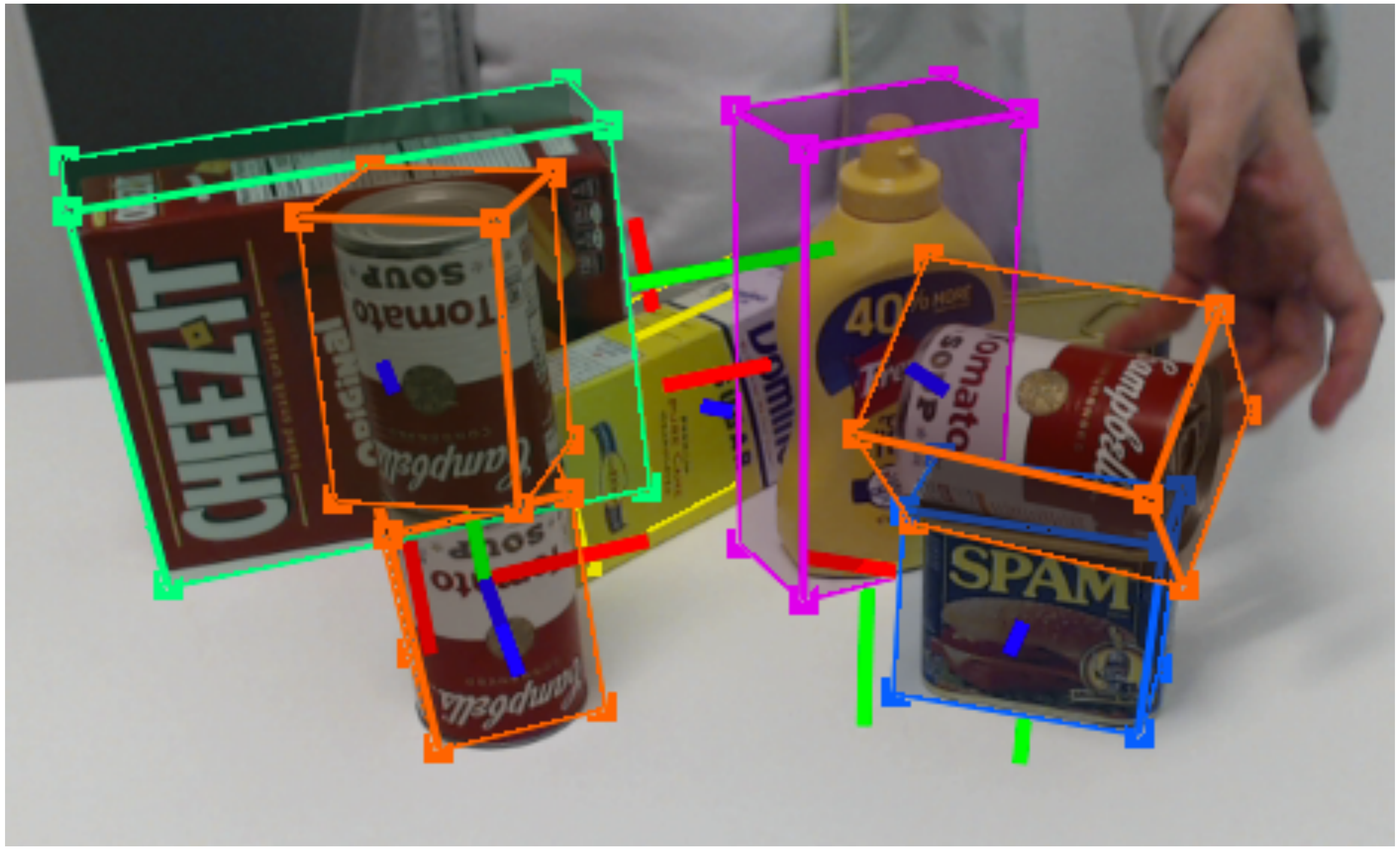
- Object Pose Estimation
- Calibrating Cameras in the Lab
- Self-Calibration

Outline

- **Object Pose Estimation**
- Calibrating Cameras in the Lab
- Self-Calibration

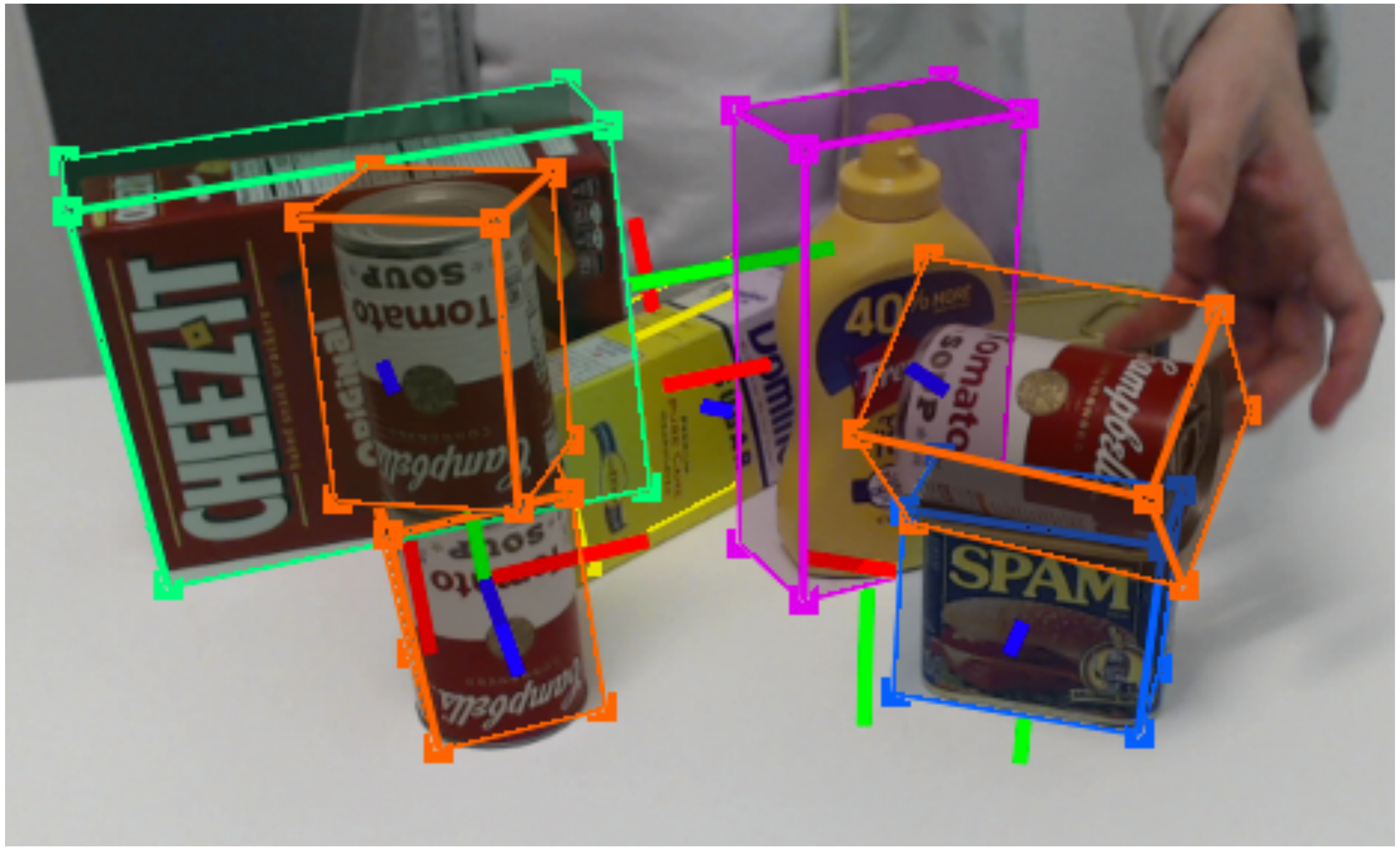
Problem Definition

- Given:
  - A 3D model of an object
  - An image of the object
- Estimate:
  - The 3D pose of the object relative to the camera



Perspective 3-Point Problem

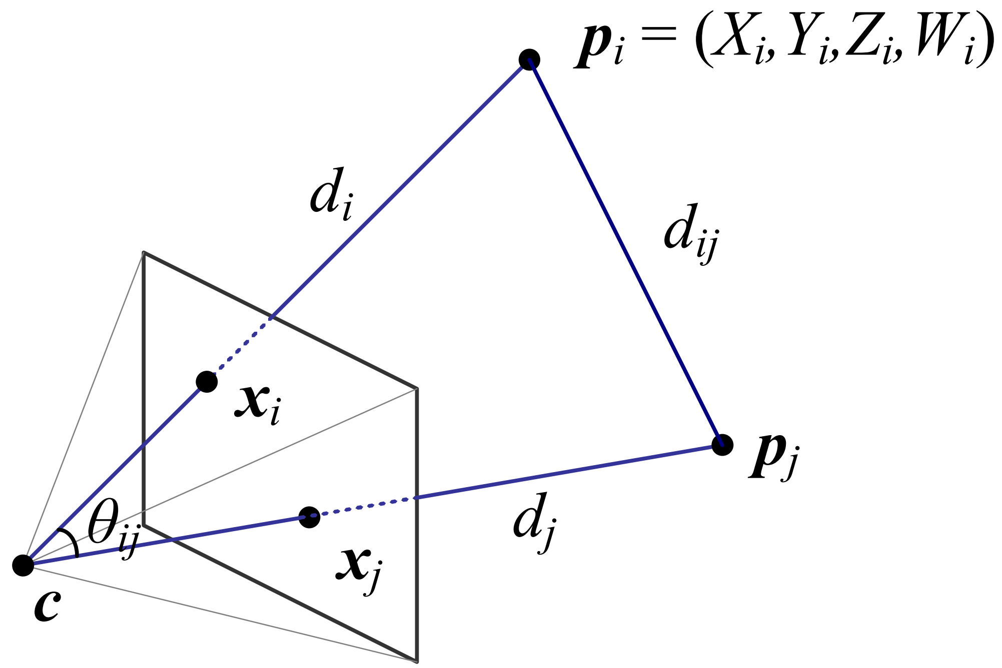
- How many degrees of freedom (parameters) are we estimating?
- How many point correspondences between 3D object and 2D image do we need?



Linear Algorithms

- 3 x 4 camera projection matrix $\boldsymbol{P}$

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \bar{\boldsymbol{p}}_w = \boldsymbol{P}\bar{\boldsymbol{p}}_w$$



$$\boldsymbol{P} = \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \end{bmatrix}$$

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

Linear Algorithms

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

- As for estimation of 2D homographies, we can form a linear estimate of the parameters p_{ij} by multiplying through by the denominator, which yields

$$\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -x_iX_i & -x_iY_i & -x_iZ_i & -x_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -y_iX_i & -y_iY_i & -y_iZ_i & -y_i \end{bmatrix} \begin{bmatrix} p_{00} \\ p_{01} \\ \vdots \\ p_{23} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- How many pairs of matching points do we need?
- Again, this estimate does not minimize the squared deviation but can be used as an initial guess for an iterative solution.
- Solve using singular value decomposition (SVD).

MATLAB function svd(A)

SVD Solution for Projection Matrix P

$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$

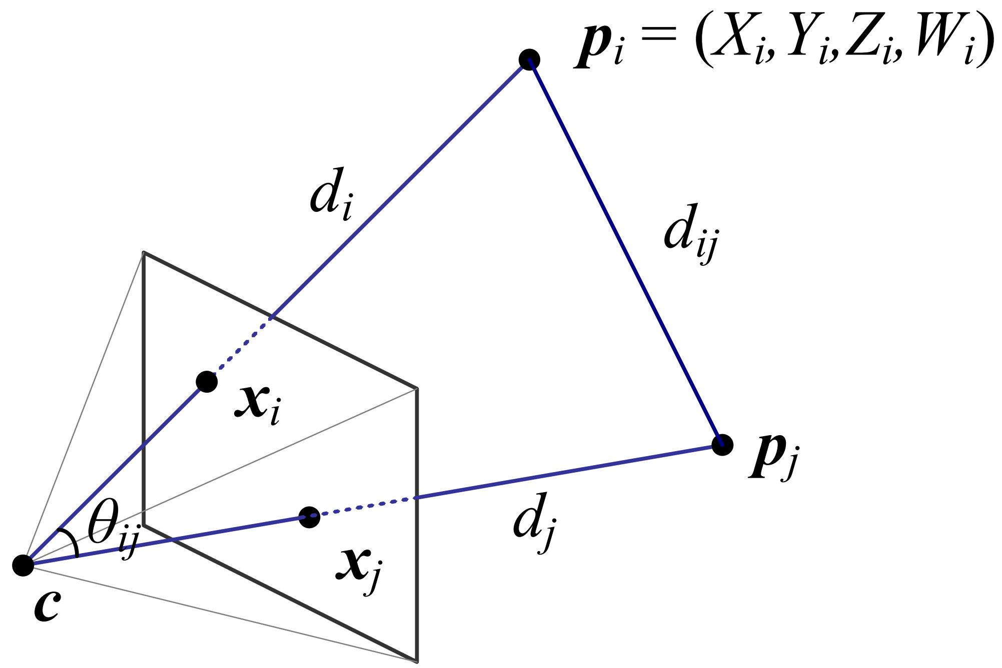
- In MATLAB: $[\boldsymbol{U},\boldsymbol{S},\boldsymbol{V}] = \mathrm{svd}(\boldsymbol{A})$
- The solution will be the last column of $\boldsymbol{V}$
- This must then be reshaped into the 3 x 4 projection matrix $\boldsymbol{P}$.
- Note that $\boldsymbol{P}$ is determined only up to a scaling constant (positive or negative).

Linear Algorithms

- 3 x 4 camera projection matrix $\boldsymbol{P}$

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \bar{\boldsymbol{p}}_w = \boldsymbol{P} \bar{\boldsymbol{p}}_w$$

$$\boldsymbol{P} = \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \end{bmatrix}$$



- Once $\boldsymbol{P}$ has been estimated, its constituents $\boldsymbol{K}$, $\boldsymbol{R}$ and $\boldsymbol{t}$ can be recovered.

- Recall that $\boldsymbol{R}$ is orthonormal and $\boldsymbol{K}$ is normally treated as upper triangular:

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

f_x and f_y: encode focal length and pixel spacing, which may be slightly different in x and y dimensions.

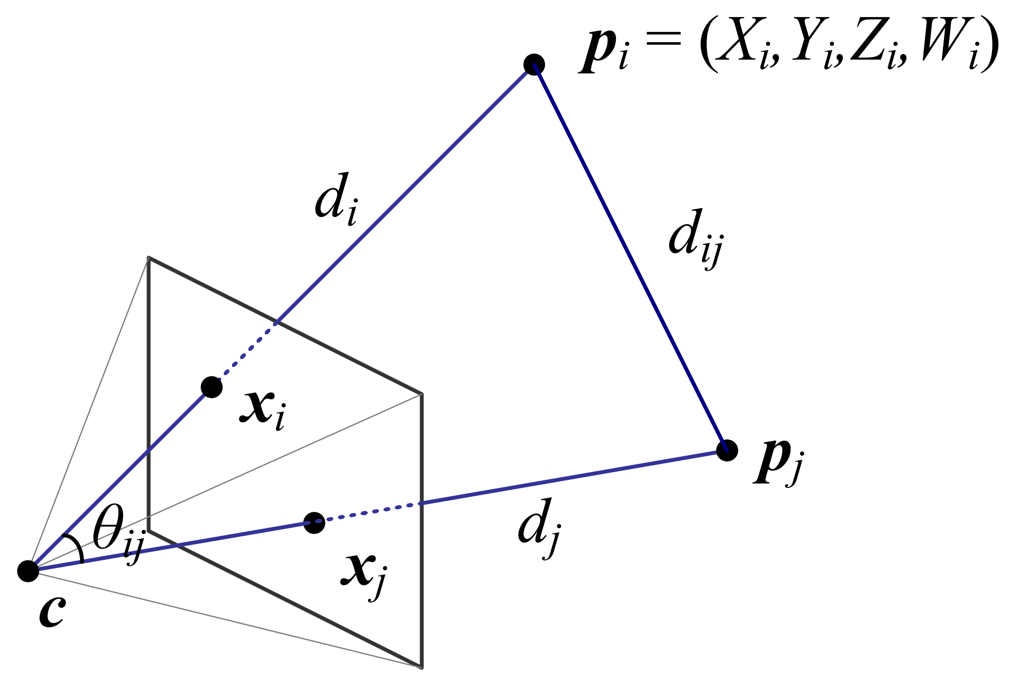
c_x and c_y: encode principal point (intersection of optic axis with sensor plane) - usually very close to centre of image

s: encodes possible skew between sensor axes (usually close to 0).

Linear Algorithms

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K}\left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]\bar{\boldsymbol{p}}_w = \boldsymbol{P}\bar{\boldsymbol{p}}_w$$

$$\boldsymbol{P} = \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \end{bmatrix} \qquad \boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$



- Thus $\boldsymbol{K}$ and $\boldsymbol{R}$ can be recovered from the first 3 columns of $\boldsymbol{P}$ using RQ decomposition.

Complexity: $O\left(MN^2 + N^3\right)$ for an $M \times N$ matrix (3×3 in our case).

MATLAB function qr(A)

Constraints on Projection Matrix *P*

Let $\boldsymbol{A} = \boldsymbol{P}(:,1:3) = \boldsymbol{KR}$

- $\boldsymbol{A} = \boldsymbol{KR} \rightarrow |\boldsymbol{A}| = |\boldsymbol{K}||\boldsymbol{R}|$
- To be a pure rotation (no reflection), $|\boldsymbol{R}| = 1$.
- K is triangular with positive diagonal elements $\rightarrow |\boldsymbol{K}| > 0$ as well.
- Thus $|\boldsymbol{A}| > 0$
- Recall that $\boldsymbol{P}$ defined up to scale factor.
- Thus if $|\boldsymbol{A}| < 0$ we multiply $\boldsymbol{P}$ by -1 so that $|\boldsymbol{A}| > 0$.

RQ Decomposition of the Projection Matrix

Let $\boldsymbol{A} = \boldsymbol{P}(:,1:3) = \boldsymbol{KR}$

- MATLAB has a QR function but no RQ function.

- To compute the RQ decomposition of $\boldsymbol{A}$ using the QR function:

$$\text{Let } M \triangleq \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

- Observations:

  - Pre-multiplication of a matrix $\boldsymbol{B}$ by $\boldsymbol{M}$ reverses the rows of $\boldsymbol{B}$ and post-multiplication reverses the columns.

  - $\boldsymbol{MM} = \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix.

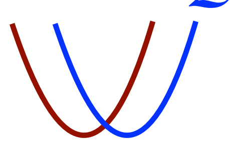
Algorithm for Computing RQ from QR

1. Compute $\tilde{\boldsymbol{A}} = \boldsymbol{MA}$
2. Compute $\tilde{\boldsymbol{Q}}\tilde{\boldsymbol{R}} = \tilde{\boldsymbol{A}}^\top$ using QR decomposition
3. Compute $\boldsymbol{Q} = \boldsymbol{M}\tilde{\boldsymbol{Q}}^\top$
4. Compute $\boldsymbol{R} = \boldsymbol{M}\tilde{\boldsymbol{R}}^\top \boldsymbol{M}$

- MATLAB code:

```
[Q,R] = qr(flipud(A)');
Q = flipud(Q');
R = flipud(fliplr(R'));
```

- $\boldsymbol{A} = \boldsymbol{KR} = \boldsymbol{RQ}$



Identifying *K*, *R* and *t*

- $\boldsymbol{A} = \boldsymbol{RQ}$
- $\boldsymbol{R}$ and $\boldsymbol{Q}$ are not uniquely defined:

Let $\boldsymbol{D}$ be a diagonal 3×3 matrix with $D_{ii} = \pm 1,\ i \in \{1,2,3\}$

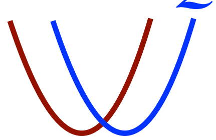
- How many distinct $\boldsymbol{D}$ matrices are there?

Let $\boldsymbol{R'} = \boldsymbol{RD}$ and $\boldsymbol{Q'} = \boldsymbol{D}^{-1}\boldsymbol{Q}$

$\boldsymbol{R'}$ is still upper diagonal and $\boldsymbol{Q'}$ is still orthonormal.

Thus $\boldsymbol{A} = \boldsymbol{R'Q'} = (\boldsymbol{RD})(\boldsymbol{D}^{-1}\boldsymbol{Q}) = \boldsymbol{RQ}$ is also a solution.

$\boldsymbol{A} = \boldsymbol{KR} = \boldsymbol{RQ}$



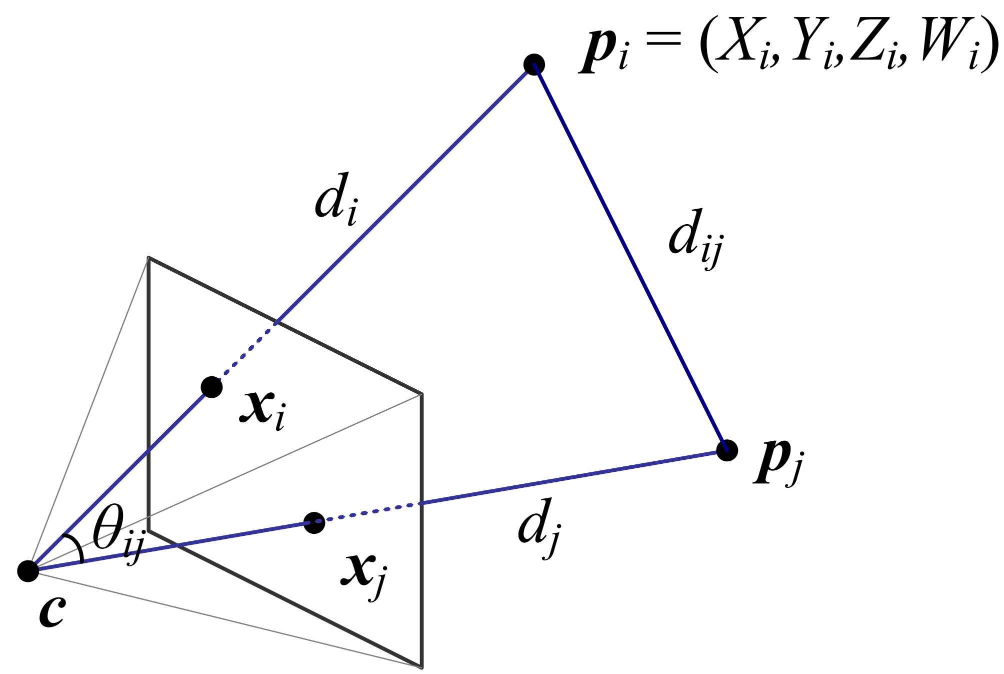
- Which of the 8 decompositions do we choose?
- Constraint: all diagonal elements of $\boldsymbol{R} = \boldsymbol{K}$ must be > 0.
  - → Set $\boldsymbol{D}_{ii} = \text{sign}(\boldsymbol{R}_{ii})$

Linear Algorithms

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \bar{\boldsymbol{p}}_w = \boldsymbol{P}\bar{\boldsymbol{p}}_w$$

$$\boldsymbol{P} = \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \end{bmatrix}$$

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$



- Given a calibrated camera ($\boldsymbol{K}$ known), $\boldsymbol{R}$ and $\boldsymbol{t}$ can be recovered with as few as 3 matched points
- Basic idea: visual angle between any pair of 2D points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the image must be the same as the visual angle between their corresponding 3D points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$.

Linear Algorithms

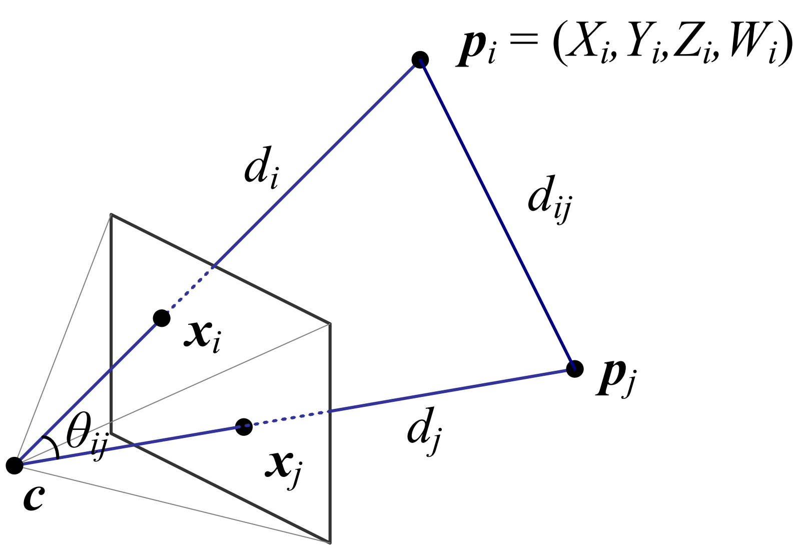
- Basic idea: visual angle between any pair of 2D points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the image must be the same as the visual angle between their corresponding 3D points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$.

  Let $\hat{\boldsymbol{x}}_i$ represent the unit vector pointing to image point $\boldsymbol{x}_i$ from the camera centre $\boldsymbol{c}$:

  $\hat{x}_i = \mathcal{N}\left(\boldsymbol{K}^{-1}\tilde{\boldsymbol{x}}_i\right) = \boldsymbol{K}^{-1}\tilde{\boldsymbol{x}}_i / \left\|\boldsymbol{K}^{-1}\tilde{\boldsymbol{x}}_i\right\|$

  the unknowns are the distances d_i from the camera origin $\boldsymbol{c}$ to the 3D points $\boldsymbol{p}_i$, where

$$\boldsymbol{p}_i = d_i \hat{\boldsymbol{x}}_i + \boldsymbol{c}$$



The cosine law for triangle $\Delta(\boldsymbol{c}, \boldsymbol{p}_i, \boldsymbol{p}_j)$ gives us

$$f_{ij}(d_i, d_j) = d_i^2 + d_j^2 - 2 d_i d_j c_{ij} - d_{ij}^2 = 0,$$

where

$$c_{ij} = \cos\theta_{ij} = \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{x}}_j$$

and

$$d_{ij}^2 = \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2.$$

Thus any triplet of constraints $f_{ij}\left(d_i, d_j\right), f_{ik}\left(d_i, d_k\right), f_{jk}\left(d_j, d_k\right)$ generates 3 equations in 3 unknowns.

End of Lecture
Nov 19, 2018

Linear Algorithms

- Two of the distances can be eliminated from the triplet of constraints to yield a quartic equation in d_i^2:

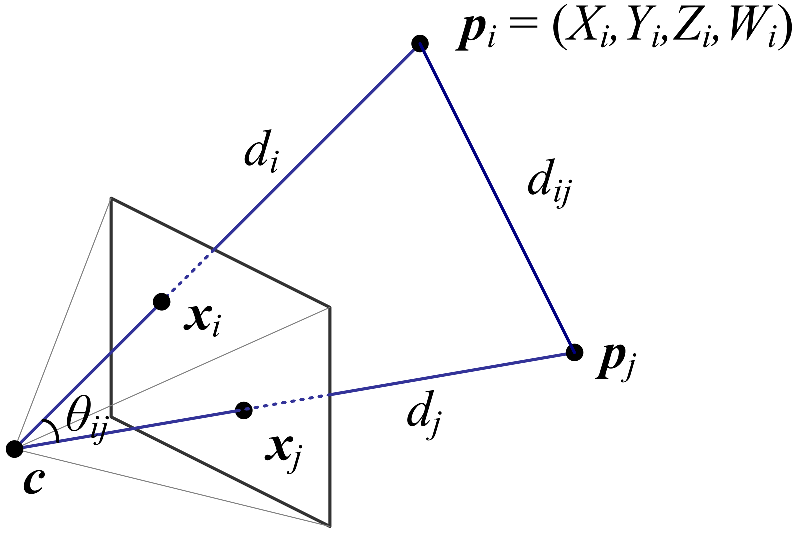
  $$a_4 d_i^8 + a_3 d_i^6 + a_2 d_i^4 + a_1 d_i^2 + a_0 = 0$$

  n point correspondences generate $(n-1)(n-2)/2$ triplets.

  pseudo-inverse can then be used to obtain estimates for $\left(d_i^8, d_i^6, d_i^4, d_i^2\right)$

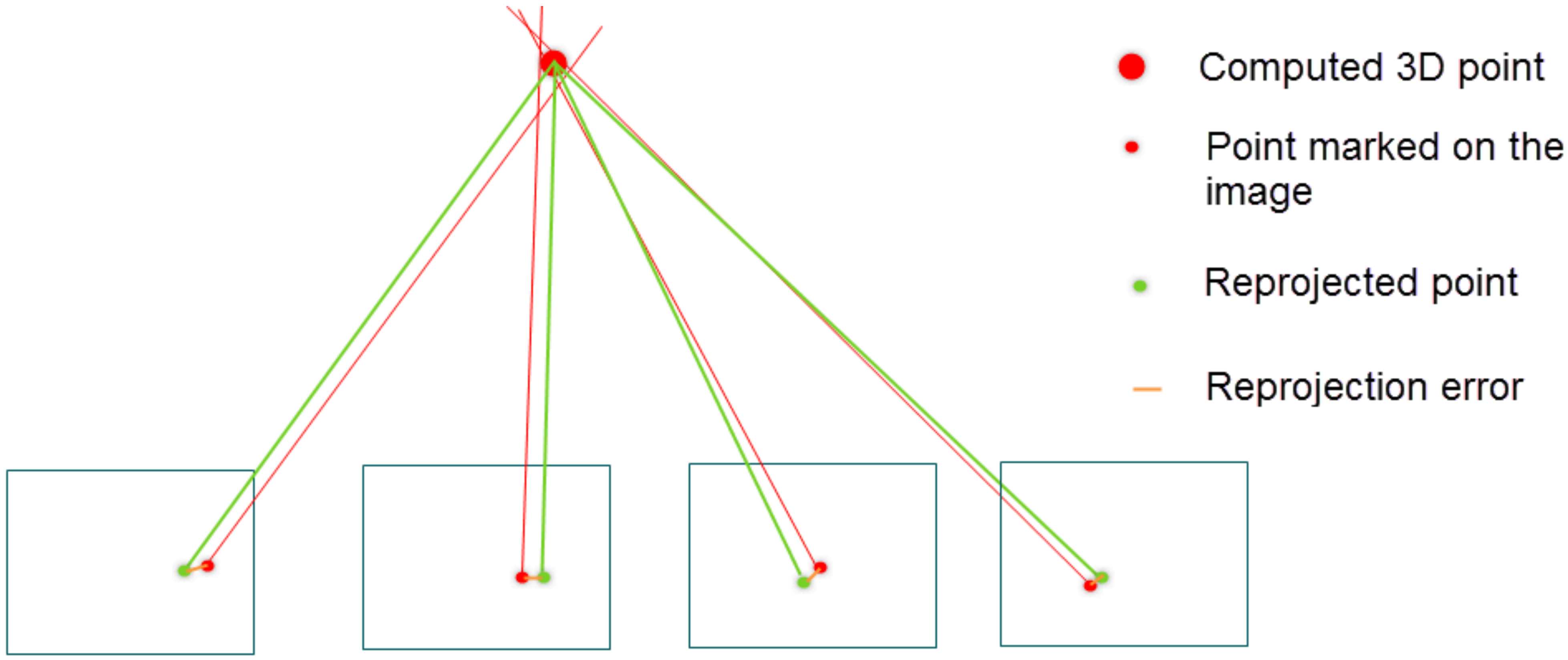
  d_i can then be estimated by averaging $\sqrt{d_i^8/d_i^6}, \sqrt{d_i^6/d_i^4}, \sqrt{d_i^4/d_i^2}, \sqrt{d_i^2}$.

- Once the d_i have been estimated, the 3D model can be aligned with the estimated 3D points $\boldsymbol{p}_i$ to estimate $\boldsymbol{R}$ and $\boldsymbol{t}$.



Iterative Algorithms

- These minimal linear one-shot algorithms have limitations:
  - Noisy (few points)
  - Do not directly minimize error
- Given these limitations, they are most useful as a means to generate an initial guess that can then be refined iteratively to minimize the **reprojection error**.
- **Definition**: **Reprojection error**
  - The deviation in the image between 2D image points $\boldsymbol{x}_i$ and their corresponding 3D points $\boldsymbol{p}_i$, projected to the image.



Iterative Algorithms

- Let $\boldsymbol{f}$ now represent projection to the image:

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{p}_i; \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{K})$$

- We now iteratively minimize a measure of the linearized reprojection error

$$E_{\mathrm{NLP}} = \sum_i \rho\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{R}}\Delta\boldsymbol{R} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{t}}\Delta\boldsymbol{t} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{K}}\Delta\boldsymbol{K} - \boldsymbol{r}_i\right),$$

where $\boldsymbol{r}_i = \hat{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_i$ is the current residual vector (2D error in predicted position)

and Sign reversed in textbook.

$\hat{\boldsymbol{x}}_i$ is the 2D image point.

$\tilde{\boldsymbol{x}}_i$ is the current estimate of the projection of 3D point $\boldsymbol{p}_i$ to the image.

Iterative Algorithms

$$E_{\mathrm{NLP}} = \sum_i \rho\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{R}}\Delta \boldsymbol{R} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{t}}\Delta \boldsymbol{t} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{K}}\Delta \boldsymbol{K} - \boldsymbol{r}_i\right),$$

- You can solve this minimization problem using MATLAB lsqnonlin.
  - If you compute the Jacobian analytically you can supply it explicitly to lsqnonlin.
  - Otherwise, lsqnonlin will compute the Jacobian numerically.
- Rotation parameters can be represented in axis/angle form
- This is the classic way to calibrate a camera (i.e., to estimate ***K***) in the lab.
- Check your solution by plotting the reprojected points!

MATLAB:
```
options.Algorithm = 'levenberg-marquardt';
p = lsqnonlin(fun,p0,[],[],options);
```

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$

Amount of rotation ($\boldsymbol{\omega}$) — Axis of rotation ($\hat{\boldsymbol{n}}$)

MATLAB:
```
r = rotationMatrixToVector(R)
```

MATLAB Implementation

Additional parameters

MATLAB:
```
err = @(p) reprojerr(p,K,x,X);
options = optimoptions('lsqnonlin','Algorithm','levenberg-marquardt');
p = lsqnonlin(err,p,[],[],options);
```

Initial guess

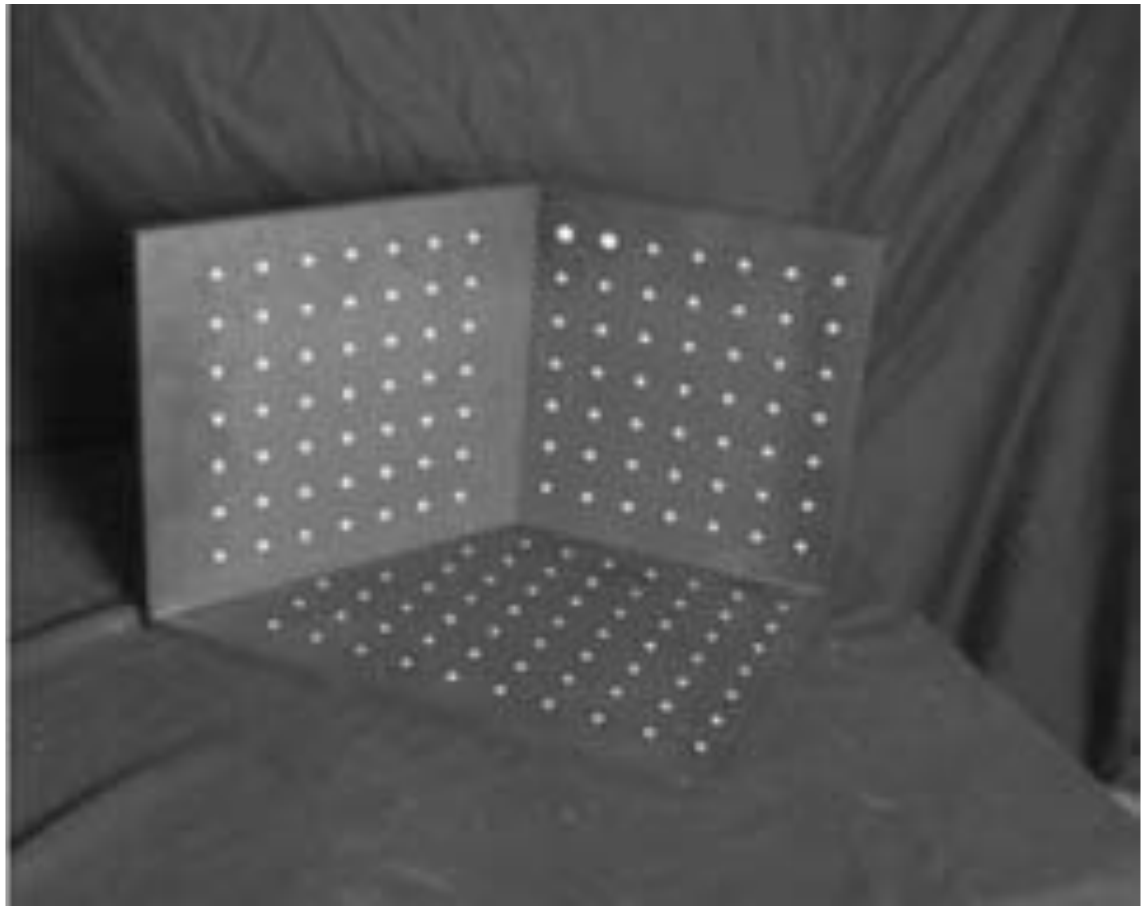
- **reprojerr** is a user-supplied function that returns a column vector of signed deviations (not squared).

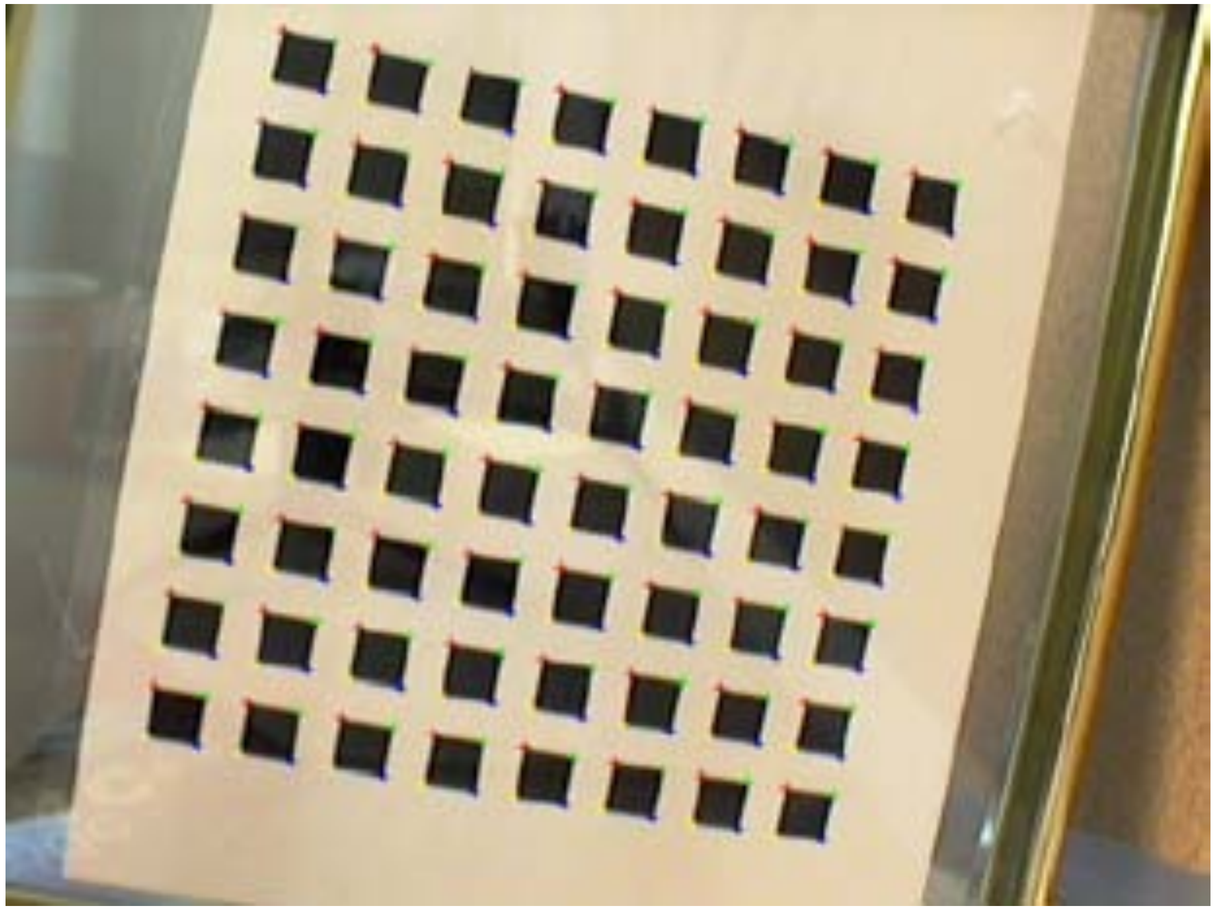
Outline

- Object Pose Estimation
- **Calibrating Cameras in the Lab**
- Self-Calibration

Calibration Pattern

- To geometrically calibrate a camera, we employ a calibration rig with known 3D dimensions.
- If the rig can be made large and placed distant from the camera, small variations in the translation of the camera will have minor impact on the image, so only ***R*** and ***K*** need to be estimated.
- However, in computer vision we commonly use smaller rigs and estimate ***t*** as well.





Standard Method (Zhang, 2000)

- In the standard approach, we capture multiple images of a planar rig from different vantages, using the camera to be calibrated.

- Correspondences ($\boldsymbol{x}_i$, $\boldsymbol{p}_i$) between the 3D keypoints on the rig and 2D image points are automatically or manually determined.

- For convenience, we employ a 3D world coordinate system anchored on the planar rig, so that the homography mapping the 3D keypoints $\boldsymbol{p}_i$ on the rig to image points $\boldsymbol{x}_i$ can be represented as

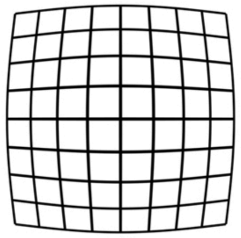
$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \overline{\boldsymbol{p}}_w = \boldsymbol{P} \overline{\boldsymbol{p}}_w \longrightarrow \overline{\boldsymbol{x}}_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \sim \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_0 & \boldsymbol{r}_1 & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \sim \tilde{\boldsymbol{H}} \overline{\boldsymbol{p}}_i$$

  where $\boldsymbol{r}_0$ and $\boldsymbol{r}_1$ are the first two columns of $\boldsymbol{R}$ and ~ represents equality up to a scaling factor.

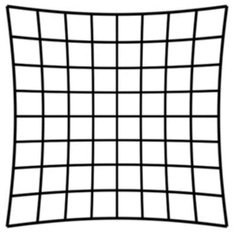
- It can be shown (Zhang, 2000) that K can be recovered from two or more images in a two-step process consisting of
  - Closed-form algebraic estimate
  - Iterative minimization of geometric (maximum likelihood) solution

Radial Distortion

- In radial distortion, points are displaced radially by an amount that increases with their distance from the image centre
  - Barrel distortion: points are displaced away from the image centre
  - Pincushion distortion: points are displaced towards the image centre



Barrel Distortion



Pincushion Distortion

- Radial distortion can be modelled by a 4th-order perturbation on these coordinates:
  - Let (x_c, y_c) be image coordinates after perspective projection but before scaling by focal length and shifting by the optical centre.

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$
$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4),$$

where $r_c^2 = x_c^2 + y_c^2$

- Optimization of the radial distortion parameters κ_1 and κ_2 can be folded into the iterative phase of the standard nonlinear camera calibration process.

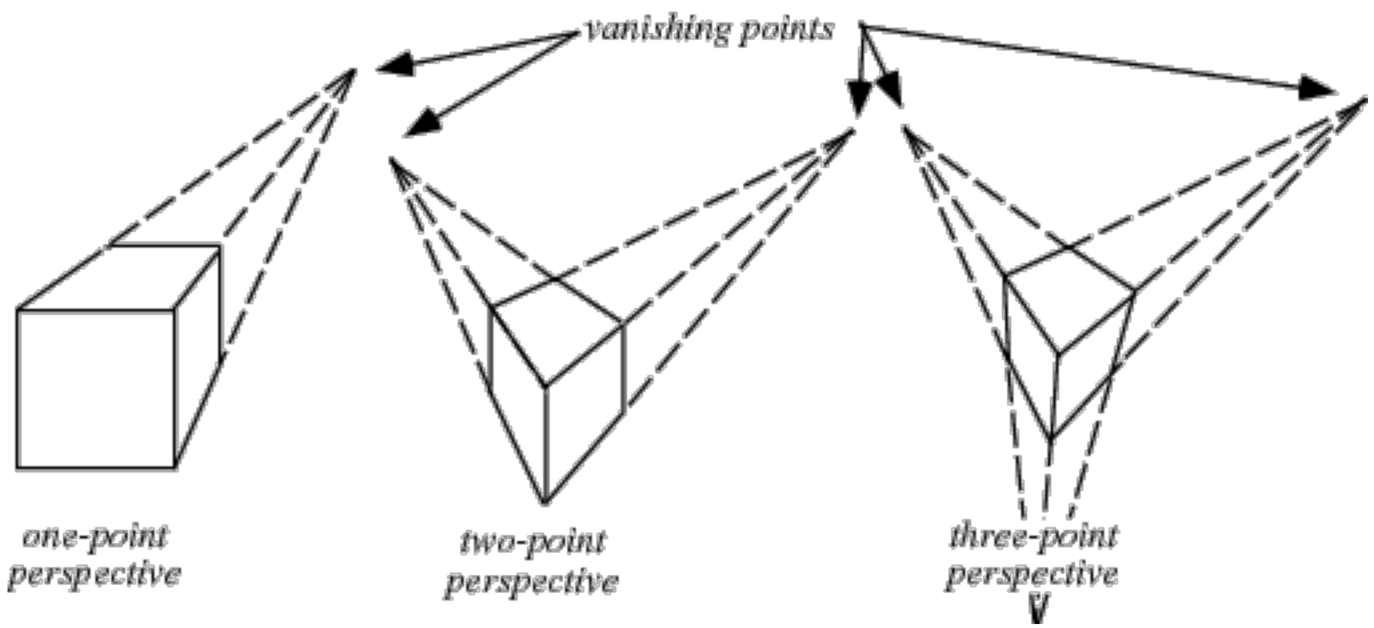
Outline

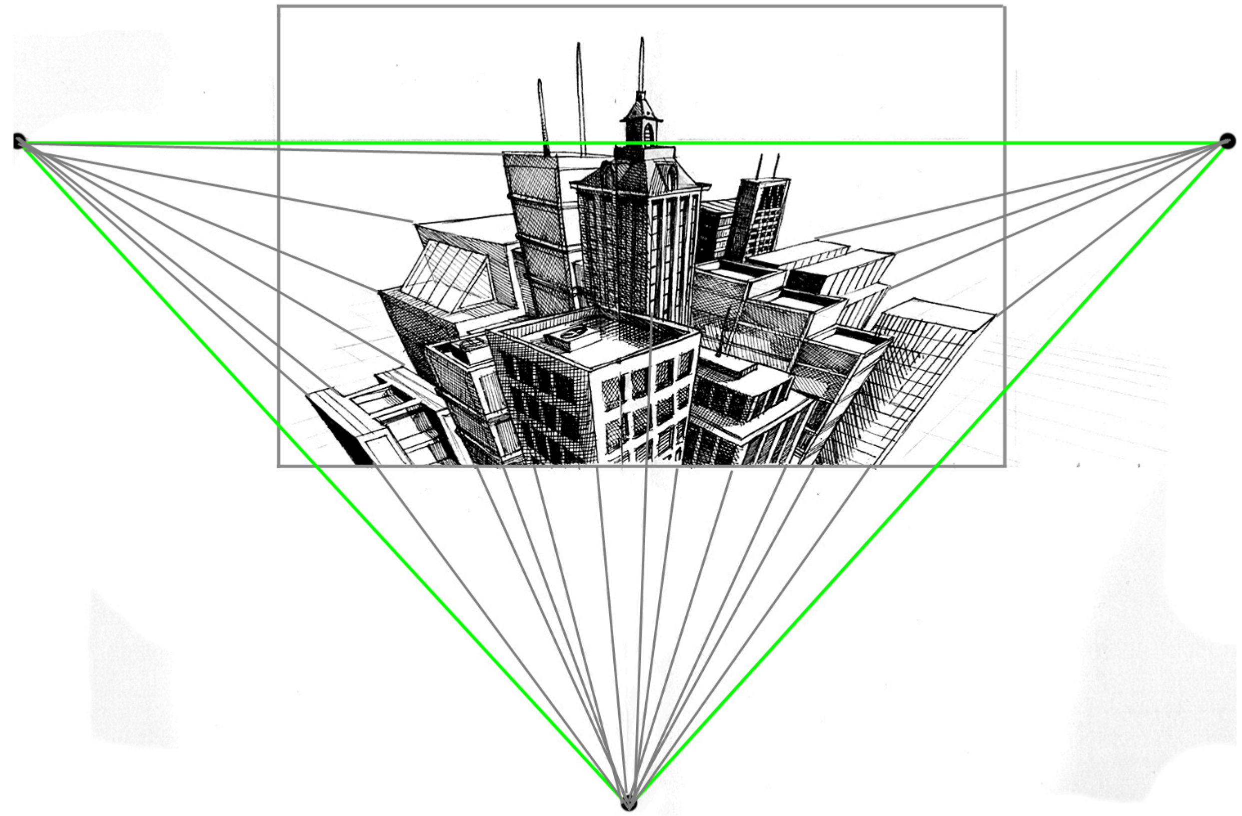
- Object Pose Estimation
- Calibrating Cameras in the Lab
- **Self-Calibration**

Self Calibration

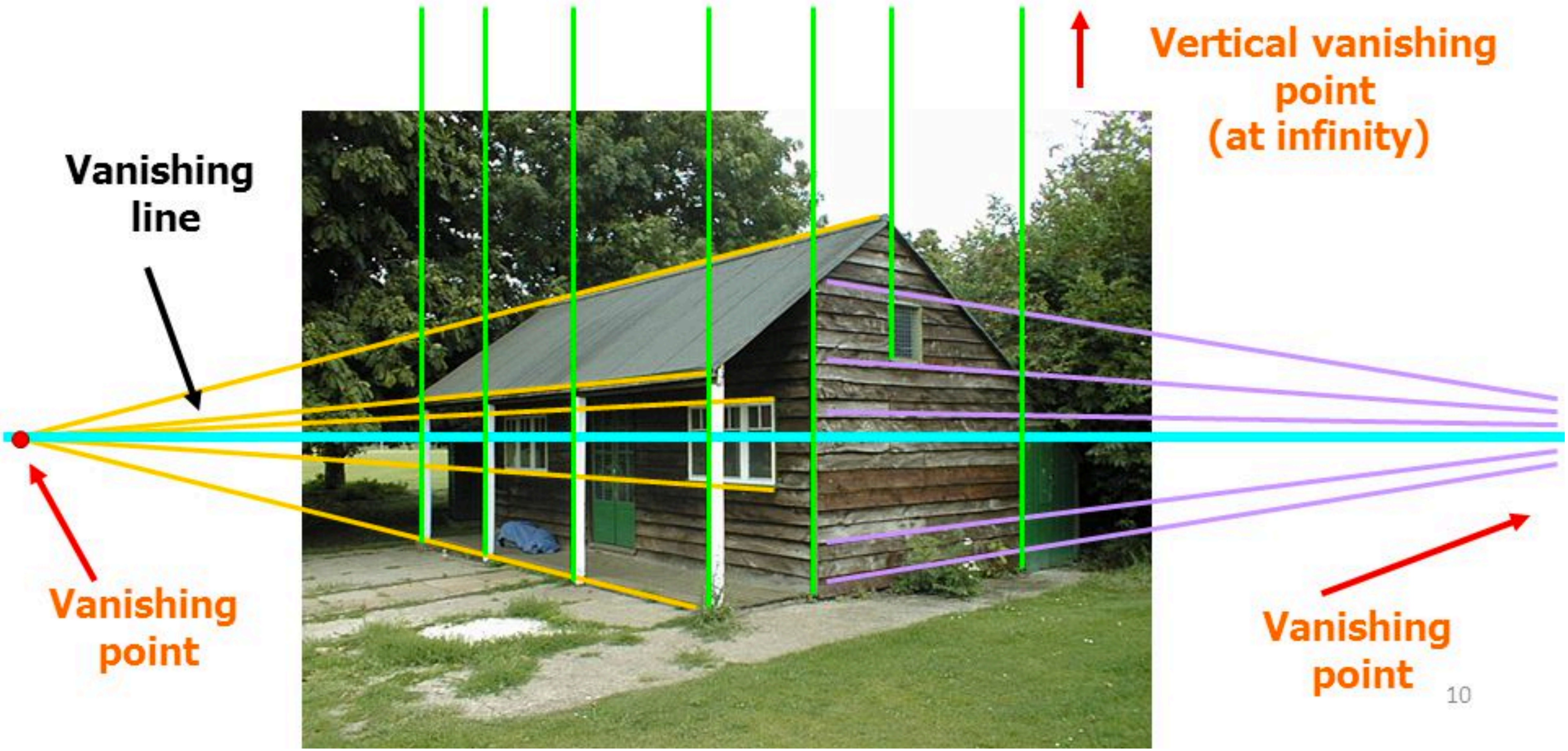
- Instead of using a known 3D calibration rig or known 3D object, can we use more general regularities of our visual world to calibrate a camera?
- One idea is to use the preponderance of parallel lines present in many visual scenes.
- A strong form of this is the Manhattan World assumption

3-Point Perspective





Vanishing Points and the Manhattan Frame



Vanishing Points

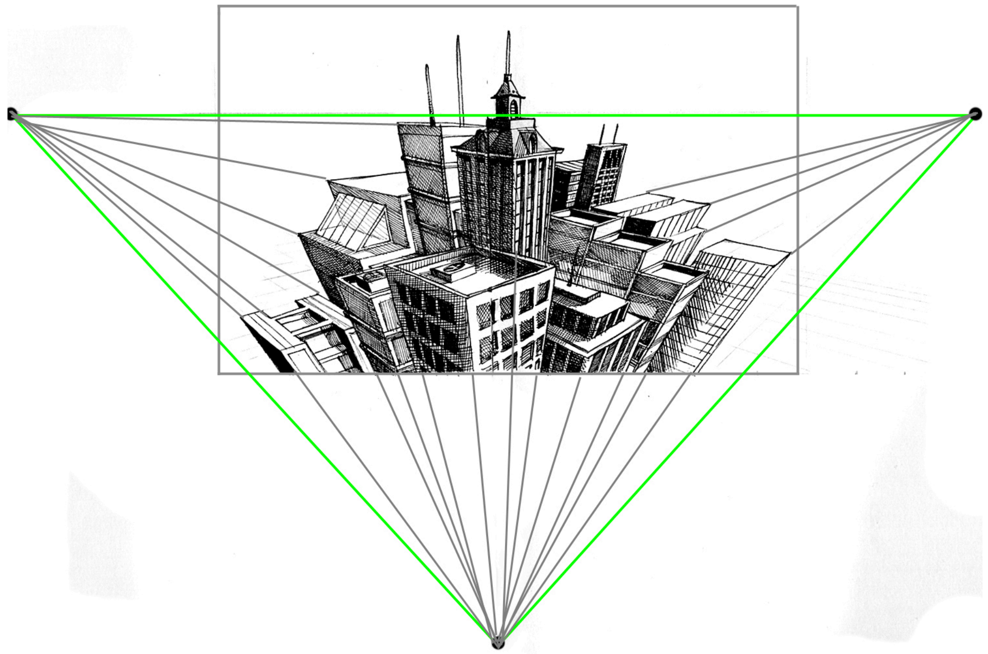
- For convenience we assume a world coordinate frame aligned with the Manhattan structure of the scene.

- Now the 3D points we know are the back-projections of the three Manhattan vanishing points, which lie at infinity along the three axes of the world frame.

- This allows us to drop the fourth column of the projection matrix $\boldsymbol{P}$, as the $[X, Y, Z]$ components of the 3D world points $\boldsymbol{p}_w$ dwarf the fourth augmented coordinate (1).

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K}\left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]\overline{\boldsymbol{p}}_w = \boldsymbol{P}\overline{\boldsymbol{p}}_w \longrightarrow \tilde{\boldsymbol{x}}_s = \boldsymbol{K}\boldsymbol{R}\boldsymbol{p}_w$$

where the $\boldsymbol{p}_w$ are simply the world axis directions (1, 0, 0), (0, 1, 0) and (0, 0, 1).

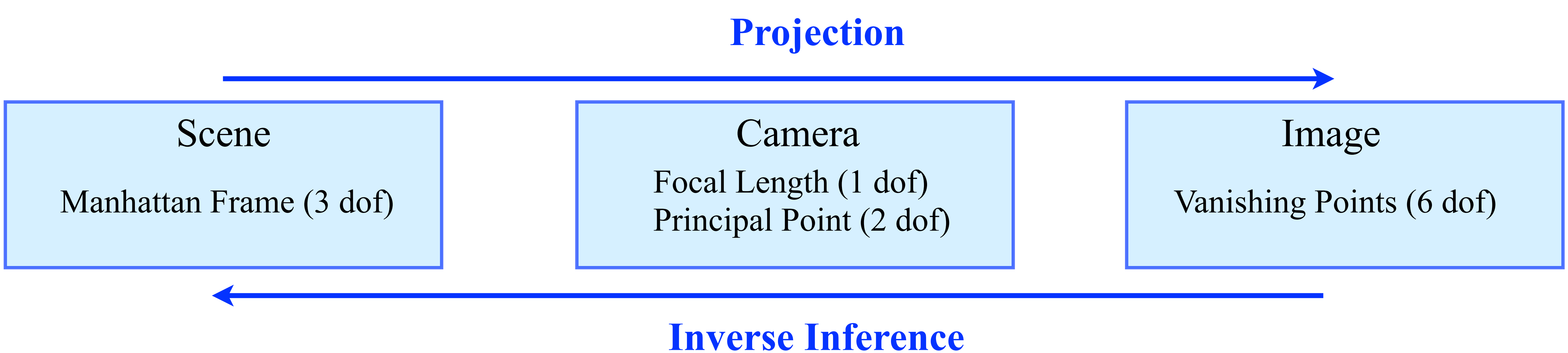


Self Calibration

- The locations of Manhattan vanishing points in the images are determined by:
  - The camera rotation (3 dof)
  - The focal length (1 dof)
  - The principal point (2 dof)

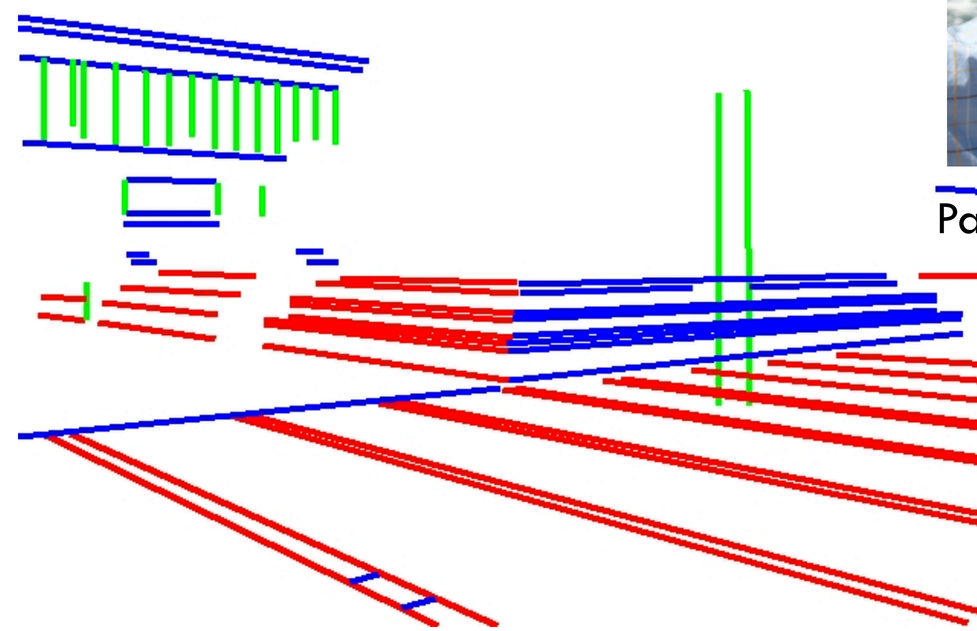
$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- If we assume a central principal point, zero skew and square pixels ($f_x = f_y$), then 2 vanishing points are in theory sufficient.
- If we have 3 vanishing points we can also estimate the principal point.



York Urban Database (2008)

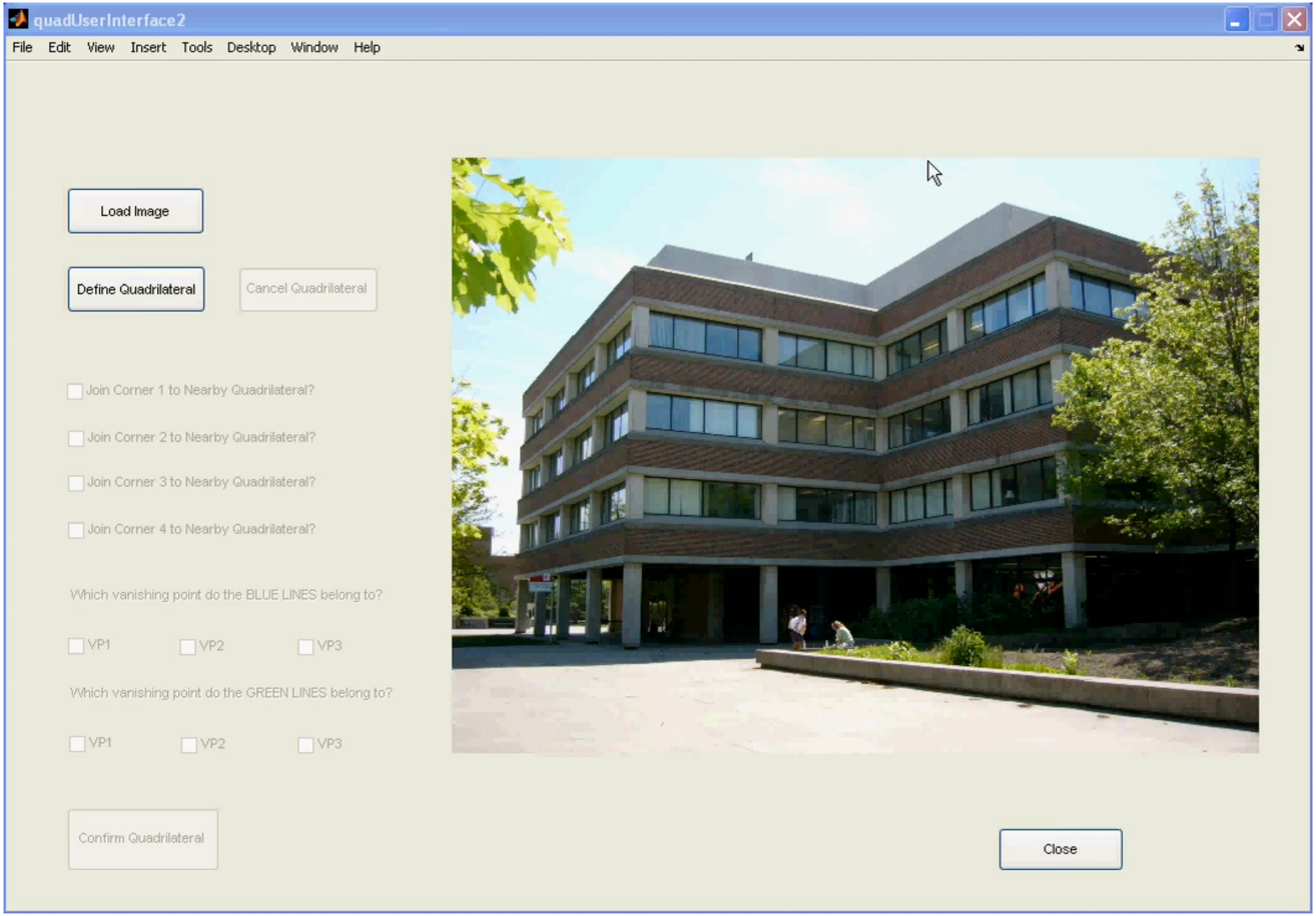
- www.elderlab.yorku.ca/YorkUrbanDB
  - 102 images of urban Toronto scenes
  - 12,122 labelled Manhattan line segments
  - Estimates of ground truth Manhattan frame for each image (estimated accuracy ~1.5 deg)



Patrick Denis

Denis, Elder & Estrada, ECCV 2008

Application: Single-View 3D Reconstruction



Patrick Denis

Denis, Elder & Estrada, ECCV 2008

Self-Calibration from Rotation

- Instead of assuming regularities in the world, we can take advantage of regularities in the motion of the camera.

- In particular, suppose we take a series of images while the camera undergoes a pure rotation about the optical centre (e.g., by spinning the camera on a tripod).

- Even though the scene is not planar, projection to the image is a 3x3 homography if we centre the world frame at the optical centre of the camera, so that translation $\boldsymbol{t}$ is always 0:

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix} \overline{\boldsymbol{p}}_w = \boldsymbol{P}\boldsymbol{p}_w \longrightarrow \tilde{\boldsymbol{x}}_s = \boldsymbol{K}\boldsymbol{R}\boldsymbol{p}_w = \tilde{\boldsymbol{H}}\boldsymbol{p}_w$$

- (From a purely rotational motion you have no way of knowing that the scene is *not* in fact planar.)

- This means that points in any pair of frames (i, j) are also related by a homography:

$$\tilde{\boldsymbol{H}}_{ij} = \boldsymbol{K}_i \boldsymbol{R}_i \boldsymbol{R}_j^{-1} \boldsymbol{K}_j^{-1} = \boldsymbol{K}_i \boldsymbol{R}_{ij} \boldsymbol{K}_j^{-1}$$

where $\boldsymbol{R}_{ij}$ is the inter-frame rotation.

Self-Calibration from Rotation

$$\tilde{\boldsymbol{H}}_{ij} = \boldsymbol{K}_i \boldsymbol{R}_i \boldsymbol{R}_j^{-1} \boldsymbol{K}_j^{-1} = \boldsymbol{K}_i \boldsymbol{R}_{ij} \boldsymbol{K}_j^{-1}$$

- We can estimate each of these homographies by identifying at least four pairs of matching points in each image.
- There are then various methods for estimating the intrinsic matrix ***K***.
- For example, assuming that ***K*** is fixed, we observe that

$$\boldsymbol{R}_{ij} \sim \boldsymbol{K}^{-1} \tilde{\boldsymbol{H}}_{ij} \boldsymbol{K} \quad \text{and} \quad \boldsymbol{R}_{ij}^{-T} \sim \boldsymbol{K}^T \tilde{\boldsymbol{H}}_{ij}^{-T} \boldsymbol{K}^{-T}$$

- and thus

$$\boldsymbol{R}_{ij} = \boldsymbol{R}_{ij}^{-T} \longrightarrow \boldsymbol{K}^{-1} \tilde{\boldsymbol{H}}_{ij} \boldsymbol{K} \sim \boldsymbol{K}^T \tilde{\boldsymbol{H}}_{ij}^{-T} \boldsymbol{K}^{-T} \longrightarrow \tilde{\boldsymbol{H}}_{ij} (\boldsymbol{K}\boldsymbol{K}^T) \sim (\boldsymbol{K}\boldsymbol{K}^T) \tilde{\boldsymbol{H}}_{ij}^{-T}$$

Self-Calibration from Rotation

- Each pair of images (i, j) introduces a set of linear constraints on $\boldsymbol{A} \triangleq \boldsymbol{K}\boldsymbol{K}^\top$

$$\tilde{\boldsymbol{H}}_{ij}(\boldsymbol{K}\boldsymbol{K}^T) \sim (\boldsymbol{K}\boldsymbol{K}^T)\tilde{\boldsymbol{H}}_{ij}^{-T}$$

- We first use SVD to solve the resulting over-constrained homogeneous system $\boldsymbol{M}\boldsymbol{a} = \boldsymbol{0}$, where $\boldsymbol{a}$ is a vector containing the six non-zero elements of $\boldsymbol{A}$.

- We then solve for $\boldsymbol{K}$ using Cholesky decomposition.

Outline

- Object Pose Estimation
- Calibrating Cameras in the Lab
- Self-Calibration