Camera model and homographies

Camera parameters; lens distortion; homography estimation

Morten R. Hannemose, mohan@dtu.dk

February 9, 2024

02504 Computer vision course lectures,
DTU Compute, Kgs. Lyngby 2800, Denmark

This lecture is being livestreamed and recorded

(hopefully)

Two feedback persons

Quiz 1 - Results

		Correct
Question 1	Line distance	67%
Question 2	Projection	79%

Add your name this week to join a leaderboard.

Learning objectives

After this lecture you should be able to:

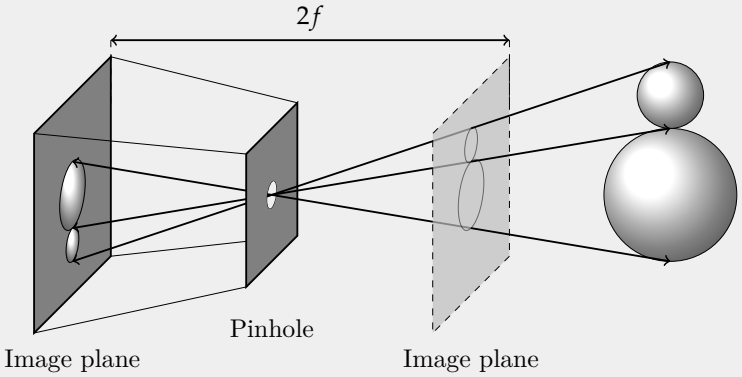
- explain the phenomelogical origin of lens distortion
- compensate for lens distortion in images
- explain the phenomelogical origin of the pinhole camera parameters
- derive the homography matrix
- explain the use of homographies in computer vision
- perform the linear estimation of a homography matrix
- use singular value decomposition (SVD) to solve linear systems

Presentation topics

- Pinhole camera
  - Intrinsics
- Lens distortion
  - Radial distortion
- Homographies
  - Application: image stitching (panorama)
- Homography estimation
  - Homogeneous least-squares solution
- Normalization of points

Pinhole camera

Recap last week



$$\boldsymbol{K} = \begin{bmatrix} f & 0 & \delta_x \\ 0 & f & \delta_y \\ 0 & 0 & 1 \end{bmatrix}$$

Camera intrinsics

- Focal length f
- Principal point $\delta x, \delta y$
- Skew parameters α and β

Focal length f

- The focal length is the distance from the pinhole to the image plane *in pixels*.
- The camera matrix makes projected coordinates equal pixels
  - δ_x and δ_y translates to make $(0, 0)$ correct
  - f performs the scaling to make the projected point into pixel coordinates

- The focal length describes the level of zoom / field of view
  - Longer focal length = more zoomed in.

Focal length f

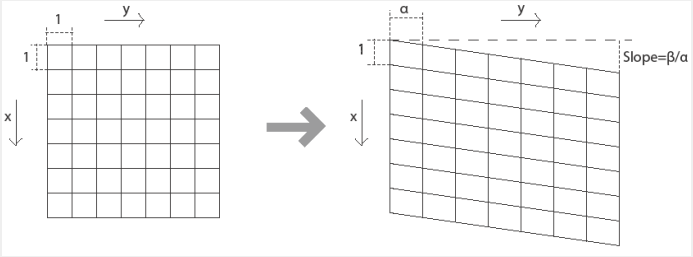
- How is the image affected visually by changing the focal length?
- Moving camera closer while zooming out to make one object have a constant size is a popular effect
- Real-life example
- Simple example
- This is called a dolly zoom / the vertigo effect

Why $\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$ **and not** $\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & f \end{bmatrix}$**?**

Skew parameters α and β

Non-square pixels (α), and non-rectangular pixels (β)

$$\boldsymbol{K} = \begin{bmatrix} f & \beta f & \delta_x \\ 0 & \alpha f & \delta_y \\ 0 & 0 & 1 \end{bmatrix}$$



Orthographic projection

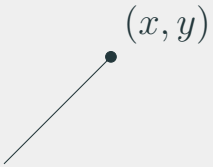
If all projection lines are parallel, then magnification $m = 1$

$$\boldsymbol{p} = \begin{bmatrix} P_x \\ P_y \end{bmatrix}$$

The pinhole model goes towards orthographic projection as the focal length goes to ∞

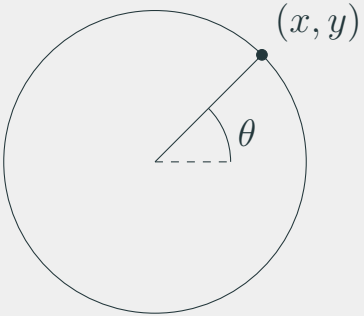
Degrees of Freedom

A point in 2D



Degrees of Freedom

A point in 2D



has one degree of freedom when it lies on a circle.

Projection matrix: Degrees of freedom

$$\boldsymbol{\mathcal{P}} = \underbrace{\begin{bmatrix} f & \beta f & \delta_x \\ 0 & \alpha f & \delta_y \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{K}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}}_{\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix}}$$

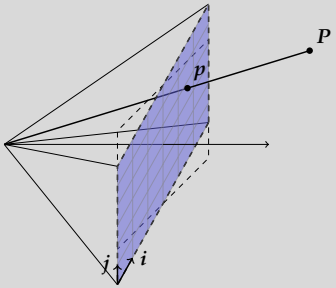
Projection matrix: Degrees of freedom

$$\boldsymbol{\mathcal{P}} = \underbrace{\begin{bmatrix} f & \beta f & \delta_x \\ 0 & \alpha f & \delta_y \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{K}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}}_{\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix}}$$

$\boldsymbol{\mathcal{P}}$ is $3 \times 4 = 12$ numbers.

$5 + 3 + 3 = 11$ degrees of freedom $+ 1$ (scale) $= 12$

Which physical properties do I need to derive the intrinsics: f, δx, δy, α, β?



Speculate; Given an unknown system, how would you estimate all parameters in $\mathcal{P}$?

Lens distortion

Help! My camera is not a pinhole camera?

Welcome to the real world!

Real lenses

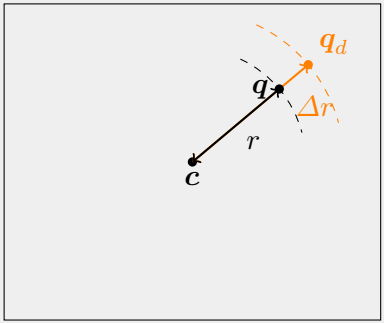
Lenses are:

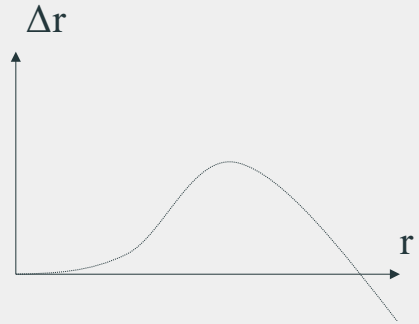
- complicated by many effects, intentional or non-intentional
- imperfect with, for example, defects
- different for each camera/lens

Almost all lenses have lens distortion!

Radial lens distortion

We distort points by a polynomial in the distance from the principal point (the radius r)





Radial distortion equations

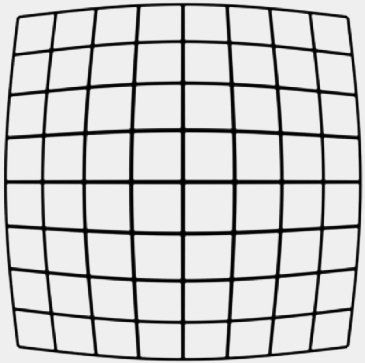
$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \mathsf{dist}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} x \\ y \end{bmatrix} \cdot \left(1 + \Delta r\left(\sqrt{x^2 + y^2}\right)\right)$$

$$\Delta r(r) = k_3 r^2 + k_5 r^4 + k_7 r^6 \ldots$$

Barrel distortion

For example:

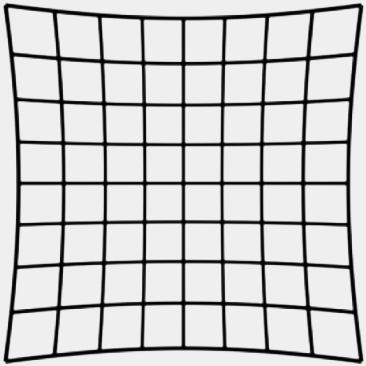
$$\Delta r(r) = -0.5r^2$$



Pincushion distortion

For example:

$$\Delta r(r) = \ 0.5r^2$$



Radial lens distortion in practice

Overfitting / extrapolation errors

$$\Delta r(r) = -0.1r^2 + \delta_{error}$$

Almost all lenses have lens distortion!

Projection under radial distortion

Projection without distortion:

$$\boldsymbol{p} = \boldsymbol{K} \underbrace{[\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{Q}}_{\boldsymbol{q}}$$

Projection under radial distortion

Projection without distortion:

$$\boldsymbol{p} = \boldsymbol{K}\underbrace{[\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{Q}}_{\boldsymbol{q}} = \boldsymbol{K}\Pi^{-1}\big(\Pi(\underbrace{[\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{Q}}_{\boldsymbol{q}})\big)$$

Projection under radial distortion

Projection without distortion:

$$\boldsymbol{p} = \boldsymbol{K}\underbrace{[\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{Q}}_{\boldsymbol{q}} = \boldsymbol{K}\Pi^{-1}\big(\Pi(\underbrace{[\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{Q}}_{\boldsymbol{q}})\big)$$

Projection with distortion:

$$\boldsymbol{p}_d = \boldsymbol{K}\underbrace{\Pi^{-1}\big(\mathrm{dist}(\Pi([\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{Q}))\big)}_{\boldsymbol{q}_d}$$

Π converts from homogeneous to inhomogeneous coordinates.
$\mathrm{dist}(\cdot)$ operates on 2D points.

Radial distortion equations [recap]

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \mathsf{dist}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} x \\ y \end{bmatrix} \cdot \left(1 + \Delta r\left(\sqrt{x^2 + y^2}\right)\right)$$

$$\Delta r(r) = k_3 r^2 + k_5 r^4 + k_7 r^6 \dots$$

Comparison to lecture notes

- This follows the math in OpenCV unlike the Lecture Notes
- Pros:
  - Projection can be done without splitting the camera matrix in two
  - Distortion parameters still work if we resize the image
- Cons:
  - Undistorting an image requires knowing the focal length

Undistortion

- You are given a distorted image, the camera matrix and the distortion coefficients
- Q1: Can you undistort points observed in the distorted image?
- Q2: Can you undistort the image?
- Discuss with the person next to you

Undistorting points (answer to Q1)

- The math given so far describes how to distort points.
- Undistorting points requires you to invert $\text{dist}(\cdot)$
- There is no analytical solution to this, so it must be done iteratively 😕

Undistorting an image (answer to Q2)

- Having the RGB-value of every (integer) pixel location ($\boldsymbol{p}$) in the undistorted image is the same as having the image.
- However, we only know the RGB-value at every location ($\boldsymbol{p}_d$) in the in the distorted image

Undistorting an image (answer to Q2)

- Having the RGB-value of every (integer) pixel location ($\boldsymbol{p}$) in the undistorted image is the same as having the image.
- However, we only know the RGB-value at every location ($\boldsymbol{p}_d$) in the in the distorted image
- Use $\text{dist}(\cdot)$ and $\boldsymbol{K}$ to find $\boldsymbol{p}_d$ from $\boldsymbol{p}$
  - $\boldsymbol{p}_d = \boldsymbol{K}\Pi^{-1}\left(\text{dist}\left(\Pi\left(\boldsymbol{K}^{-1}\boldsymbol{p}\right)\right)\right)$
- Use bilinear interpolation to compute the RGB-value in the distorted image at the (non-integer) point $\boldsymbol{p}_d$.
- Not necessary to invert the distortion function!

Questions? Short break

Homographies

Practical example

- No volunteers needed 😉

Practical example

- No volunteers needed 😉
- Knowing that a point seen with a camera lies on a plane lets us know exactly where on the plane it is.

The homography

Consider the following plane:

$$\begin{aligned}\boldsymbol{P} &= a\boldsymbol{A} + b\boldsymbol{B} + \boldsymbol{C}\,, \\ &= \begin{bmatrix}\boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C}\end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}\,.\end{aligned}$$

$\boldsymbol{C}$ is a point on the plane, and $\boldsymbol{A}$ and $\boldsymbol{B}$ are vectors that span the plane.

The homography

Projections of points on a plane:

$$\begin{aligned} \boldsymbol{q} &= \boldsymbol{\mathcal{P}} \boldsymbol{P}_h \,, \\ &= \boldsymbol{\mathcal{P}} \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} , \end{aligned}$$

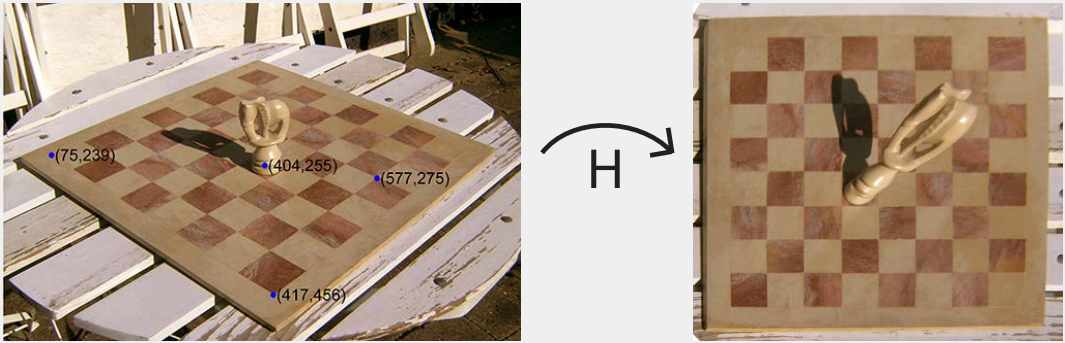
The homography

Projections of points on a plane:

$$\begin{aligned} \boldsymbol{q} &= \boldsymbol{\mathcal{P}}\boldsymbol{P}_h\,, \\ &= \underbrace{\boldsymbol{\mathcal{P}}\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{H}} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}, \end{aligned}$$

$\boldsymbol{H}$ is a 3×3 matrix called a homography.

The homography



Homographies between images

Two cameras photographing the same plane let's us establish pixel correspondence:

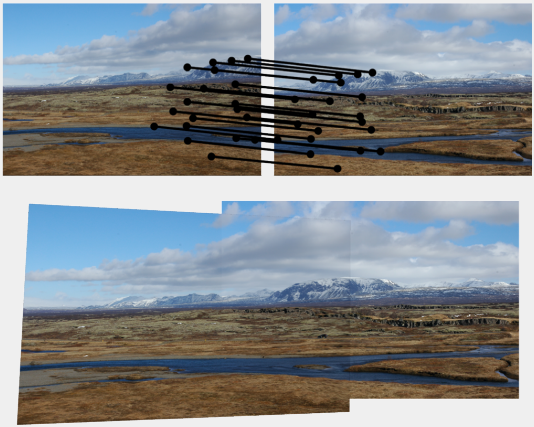
$$\boldsymbol{q}_1 = \boldsymbol{H}_1\boldsymbol{Q}, \quad \boldsymbol{q}_2 = \boldsymbol{H}_2\boldsymbol{Q}$$

$$\begin{aligned} \boldsymbol{q}_1 &= \boldsymbol{H}_1\boldsymbol{H}_2^{-1}\boldsymbol{q}_2 \\ &= \boldsymbol{H}\boldsymbol{q}_2 \end{aligned}$$

Two view geometry without baseline

If two cameras are at the same point we can use a homography to relate their pixels

Image stitching



You get to implement this in week 9.

Homography multiplication

If you multiply a homography $\boldsymbol{H}$ by a scalar, how does that change the mapping it describes?

Homography multiplication

If you multiply a homography $\boldsymbol{H}$ by a scalar, how does that change the mapping it describes?

It doesn't!

$$\boldsymbol{q}_1 = \boldsymbol{H}\boldsymbol{q}_2$$
$$s\boldsymbol{q}_1 = (s\boldsymbol{H})\boldsymbol{q}_2$$

$\boldsymbol{q}_1$ and $s\boldsymbol{q}_1$ are the same point.

Homography degrees of freedom

- 3×3 matrix has 9 numbers
- Scale is arbitrary
  - A homography has 8 degrees of freedom
- We need 4 pairs of points to estimate a homography
  - Each point pair imposes two constraints (x and y)

Any two images of the same plane are related by a homography

Homography estimation

Homography estimation

Consider a number of points $\boldsymbol{q}_{1i}$ and $\boldsymbol{q}_{2i}$. The homography relates them as follows:

$$\boldsymbol{q}_{1i} = \boldsymbol{H}\boldsymbol{q}_{2i}$$

Homography rewritten i

The cross product of two parallel vectors is the zero vector:

$$\begin{aligned} \boldsymbol{q}_{1i} &= \boldsymbol{H}\boldsymbol{q}_{2i} \Rightarrow \\ \mathbf{0} &= \boldsymbol{q}_{1i} \times \boldsymbol{H}\boldsymbol{q}_{2i}\,, \end{aligned}$$

we can isolate $\boldsymbol{H}$ in this expression ...

Homography rewritten ii

$$
\begin{aligned}
\mathbf{0} &= \boldsymbol{q}_{1i} \times \boldsymbol{H}\boldsymbol{q}_{2i}\,, \\
&= \boldsymbol{B}^{(i)}\mathsf{flatten}(\boldsymbol{H}^{\mathsf{T}})\,, \\
\boldsymbol{B}^{(i)} &= \begin{bmatrix} 0 & -x_{2i} & x_{2i}y_{1i} & 0 & -y_{2i} & y_{2i}y_{1i} & 0 & -1 & y_{1i} \\ x_{2i} & 0 & -x_{2i}x_{1i} & y_{2i} & 0 & -y_{2i}x_{1i} & 1 & 0 & -x_{1i} \\ -x_{2i}y_{1i} & x_{2i}x_{1i} & 0 & -y_{2i}y_{1i} & y_{2i}x_{1i} & 0 & -y_{1i} & x_{1i} & 0 \end{bmatrix}, \\
\mathsf{flatten}(\boldsymbol{H}^{\mathsf{T}}) &= \begin{bmatrix} H_{11} & H_{21} & H_{31} & H_{12} & H_{22} & H_{32} & H_{13} & H_{23} & H_{33} \end{bmatrix}^{\mathsf{T}}
\end{aligned}
$$

Defining $\boldsymbol{B}^{(i)}$

The $\boldsymbol{B}^{(i)}$ matrices can be implemented with the Kronecker product `np.kron`

$$\begin{aligned}\boldsymbol{B}^{(i)} &= \boldsymbol{q}_{2i}^{\mathsf{T}} \otimes [\boldsymbol{q}_{1i}]_{\times} \\ &= \boldsymbol{q}_{2i}^{\mathsf{T}} \otimes \begin{bmatrix} 0 & -1 & y_{1i} \\ 1 & 0 & -x_{1i} \\ -y_{1i} & x_{1i} & 0 \end{bmatrix}\end{aligned}$$

Homography matrix solution

Given n corresponding points, we stack their respective $\boldsymbol{B}^{(i)}$ matrices into a tall matrix $\boldsymbol{B}$:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}^{(1)} \\ \boldsymbol{B}^{(2)} \\ \vdots \\ \boldsymbol{B}^{(n)} \end{bmatrix}$$

$$\mathbf{0} = \boldsymbol{B}\ \text{flatten}(\boldsymbol{H}^{\mathsf{T}})$$

How do we solve this linear system of equations?

Homogeneous least-squares problem

- Problem of type: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$
- When noise is present, cannot be satisfied exactly
- Equivalent to $\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x}\|_2^2$
- Trivial solution $\boldsymbol{x} = \boldsymbol{0}$ (not useful)
- $\boldsymbol{x}$ can be arbitrarily scaled
  - We impose $||\boldsymbol{x}||_2 = 1$

Homogeneous least-squares solution

Rewrite problem

$$\begin{aligned} \|\boldsymbol{A}\boldsymbol{x}\|_2^2 = \\ (\boldsymbol{A}\boldsymbol{x})^\mathsf{T}\boldsymbol{A}\boldsymbol{x} = \\ \boldsymbol{x}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} \end{aligned}$$

Assume $\boldsymbol{x}$ is an eigenvector of the square matrix $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$.

Homogeneous least-squares solution

Which eigenvector?

Eigen-decomposition of $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$:

$$\boldsymbol{A}^\mathsf{T}\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T},$$
$$\boldsymbol{v}_i^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{v}_i = \underbrace{\boldsymbol{v}_i^\mathsf{T}\boldsymbol{V}}_{e_i^\mathsf{T}}\,\boldsymbol{\Lambda}\,\underbrace{\boldsymbol{V}^\mathsf{T}\boldsymbol{v}_i}_{e_i} = \lambda_i\,,$$

where e_i is the vector with 0 everywhere except for the i^{th} index.

To minimize λ_i we choose $\boldsymbol{v}_i$ (and $\boldsymbol{x}$) as the eigenvector with the smallest eigenvalue.

Least-squares solution with SVD

We can use the SVD (`np.linalg.svd`)

$$\begin{aligned} \boldsymbol{A}^\mathsf{T}\boldsymbol{A} &= \left(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\mathsf{T}\right)^\mathsf{T}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\mathsf{T} \\ &= \boldsymbol{V}\boldsymbol{\Sigma}^\mathsf{T}\boldsymbol{U}^\mathsf{T}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\mathsf{T} \\ &= \boldsymbol{V}\underbrace{\boldsymbol{\Sigma}^\mathsf{T}\boldsymbol{\Sigma}}_{\boldsymbol{\Lambda}}\boldsymbol{V}^\mathsf{T} \end{aligned}$$

$\boldsymbol{V}$ are the eigenvectors of $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$, and the minimum solution has the smallest singular value.

Least-squares solution with SVD

- When we use the SVD, we avoid computing $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$
- The least-squares solution is
  - the right singular vector of $\boldsymbol{A}$ corresponding the smallest singular value
  - the eigenvector of $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$ corresponding to the smallest eigenvalue

The solution of $Ax = 0$ where $||x||_2 = 1$ is found using SVD.

$x = v$ where the right singular vector v corresponds to the smallest singular value.

Normalization of points

Numerical instability

Experience has shown that most of these algorithms can be numerically unstable.

What can we do? We can normalize!

Point normalization

Consider the collection of n points $\boldsymbol{p}_i$ that has has mean and standard deviation as follows

$$\text{mean}(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n) = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} = \boldsymbol{\mu} \text{ and}$$

$$\text{std}(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n) = \begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = \boldsymbol{\sigma}$$

Point normalization

We would like to normalize these points so they have mean $\mathbf{0}$ and standard deviation $\mathbf{1}$.
Let us call the normalized points $\tilde{\boldsymbol{p}}_i$.

$$\text{mean}(\tilde{\boldsymbol{p}}_1, \tilde{\boldsymbol{p}}_2, \ldots, \tilde{\boldsymbol{p}}_n) = \mathbf{0} \text{ and}$$
$$\text{std}(\tilde{\boldsymbol{p}}_1, \tilde{\boldsymbol{p}}_2, \ldots, \tilde{\boldsymbol{p}}_n) = \mathbf{1}$$

Point normalization

The relation between $\boldsymbol{p}_i$ and $\tilde{\boldsymbol{p}}_i$ is then

$$\tilde{\boldsymbol{p}}_i = \frac{\boldsymbol{p}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \Leftrightarrow$$
$$\boldsymbol{p}_i = \boldsymbol{\sigma}\tilde{\boldsymbol{p}}_i + \boldsymbol{\mu}$$

Can we write this with homogeneous coordinates with some transformation $\boldsymbol{T}$?

$$\tilde{\boldsymbol{q}}_i = \boldsymbol{T}\boldsymbol{q}_i$$

Point normalization in homogeneous coordinates

$$\tilde{\boldsymbol{q}}_i = \underbrace{\begin{bmatrix} 1/\sigma_x & 0 & -\mu_x/\sigma_x \\ 0 & 1/\sigma_y & -\mu_y/\sigma_y \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{T}} \boldsymbol{q}_i$$

$$\boldsymbol{q}_i = \underbrace{\begin{bmatrix} \sigma_x & 0 & \mu_x \\ 0 & \sigma_y & \mu_y \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{T}^{-1}} \tilde{\boldsymbol{q}}_i$$

It is easiest to implement $\boldsymbol{T}^{-1}$ and invert it to obtain $\boldsymbol{T}$.

Point normalization in linear algorithms

Now we can estimate the homography using $\tilde{\boldsymbol{p}}_{1i}$ and $\tilde{\boldsymbol{p}}_{2i}$ and obtain the homography $\widetilde{\boldsymbol{H}}$.

How do we get the homography $\boldsymbol{H}$ that operates on the original points?

$$
\begin{aligned}
\tilde{\boldsymbol{q}}_{1i} &= \widetilde{\boldsymbol{H}}\tilde{\boldsymbol{q}}_{2i} = \\
\boldsymbol{T}_1\boldsymbol{q}_{1i} &= \widetilde{\boldsymbol{H}}\boldsymbol{T}_2\boldsymbol{q}_{2i} \\
\boldsymbol{q}_{1i} &= \underbrace{\boldsymbol{T}_1^{-1}\widetilde{\boldsymbol{H}}\boldsymbol{T}_2}_{\boldsymbol{H}}\boldsymbol{q}_{2i}
\end{aligned}
$$

Exercise tips

- You make mistakes and you learn from them.
- Follow these good practice tips:
- `im = cv2.imread('im.jpg')[:,:,::-1]`

Exercise tips

- You make mistakes and you learn from them.
- Follow these good practice tips:
- `im = cv2.imread('im.jpg')[:,:,::-1]`
- `im = im.astype(float)/255`

Learning objectives

After this lecture you should be able to:

- explain the phenomelogical origin of lens distortion
- compensate for lens distortion in images
- explain the phenomelogical origin of the pinhole camera parameters
- derive the homography matrix
- explain the use of homographies in computer vision
- perform the linear estimation of a homography matrix
- use singular value decomposition (SVD) to solve linear systems

Exercise time!