Introduction to Computer Vision

Course overview

Morten R. Hannemose, mohan@dtu.dk

February 3, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Welcome

About me: Morten Rieger Hannemose
Assistant prof. at DTU Compute, Section for Visual Computing
Background in computer vision and differentiable rendering

I work with various applications within computer vision
camera calibration, 3D scanning, and skin cancer.

What is computer vision?

Making machines see and perceive

Observing requires a lot more than *just* taking a photo!

- Seeing with cameras
- Recognize features
- Measure lengths and depth

Core elements of this course

This course focuses on the 3D aspects of computer vision.

- Camera model
- Multi view geometry
- 3D reconstruction
- Image features and matching

Other courses

Prerequisite: 02502 Image analysis

- Digital representation of images
- Image filters
- Image features

Other courses in computer vision/image analysis:

- 02506 Advanced Image Analysis
- 02514 Deep Learning in Computer Vision

Course overview

- 12 weeks of lectures + 1 guest lecture
  - Shorter lecture followed by examples and exercises
- Written exam
  - 4 hours
  - Multiple choice

Reading material

- "Computer Vision: Algorithms and Applications" by R. Szelisky, 2nd ed. (F&P)
- Lecture notes by Henrik Aanæs (LN)
- Two papers: Zhang 2000 and Zhang 2018
- Further reading:
  - "Multiple view geometry in computer vision", by R. Hartley and A. Zisserman

Exercises

- Find them on DTU Learn
- No hand-ins, some solutions
- Help each other and use the TAs
  - Don't expect TAs to magically spot errors
  - Learn how to debug from them

Questions?

- Feedback is welcome throughout the course

Pinhole camera

and homogeneous coordinates

Morten R. Hannemose, mohan@dtu.dk

February 3, 2023

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

Learning objectives

After this lecture you should be able to:

- explain homogeneous coordinates
- convert to and from homogeneous coordinates
- perform relevant coordinate transformations
- explain the pinhole camera model

Presentation topics

- Pinhole camera
- Perspective transformations
- Euclidean (rigid) transformations
- Homogeneous coordinates
  - Lines in homogenous coordinates
  - Summary of homogeneous coordinates
- Pinhole camera model
  - Intrinsics
  - Extrinsics

Pinhole camera

What is a “good” camera model?

. . . in terms of accuracy vs. ease-of-use?

1. As accurate as possible
2. As easy to use as possible
3. somewhere between 1 and 2

Pinhole camera

The projected image appears upside down

Each point in an image corresponds to a direction from the camera

Real life example

Can I get two volunteers?

Real life example

Can I get two volunteers?

- A point seen in a single camera must be along a specific line in the other camera.
- Seeing the same point in two cameras is enough to find the point in 3D.

Perspective transformations

Pinhole camera again

When modelling we place the "image plane" in front of the camera.
The distance from image plane to camera is the focal length f.

Perspective projection

$$\boldsymbol{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix},$$

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

Euclidean (rigid) transformations

Euclidean (rigid) transformations: rotations and translations

Rotation matrix $\boldsymbol{R}$ and translation vector $\boldsymbol{t}$

$$\boldsymbol{P}_1 = \boldsymbol{t} + \boldsymbol{R}\boldsymbol{P}_0$$

Robot arm transformations

$$\boldsymbol{P}_1 = \boldsymbol{R}_1 \boldsymbol{P}_0 + \boldsymbol{t}_1$$

$$\boldsymbol{P}_2 = \boldsymbol{R}_2 \boldsymbol{P}_1 + \boldsymbol{t}_2$$

$$\boldsymbol{P}_3 = \boldsymbol{R}_3 \boldsymbol{P}_2 + \boldsymbol{t}_3$$

$$\boldsymbol{P}_4 = \boldsymbol{R}_4 \boldsymbol{P}_3 + \boldsymbol{t}_4$$

Robot arm transformations

$$\boldsymbol{P}_4 = \boldsymbol{R}_4(\boldsymbol{R}_3(\boldsymbol{R}_2(\boldsymbol{R}_1\boldsymbol{P}_0 + \boldsymbol{t}_1) + \boldsymbol{t}_2) + \boldsymbol{t}_3) + \boldsymbol{t}_4$$

$$\boldsymbol{P}_4 = \boldsymbol{R}_4\boldsymbol{R}_3\boldsymbol{R}_2\boldsymbol{R}_1\boldsymbol{P}_0 + \boldsymbol{R}_4\boldsymbol{R}_3\boldsymbol{R}_2\boldsymbol{t}_1 + \boldsymbol{R}_4\boldsymbol{t}_3 + \boldsymbol{R}_4\boldsymbol{R}_3\boldsymbol{t}_2 + \boldsymbol{t}_4$$

Homogeneous coordinates

Homogeneous coordinates

Here is a point in 3D:

$$\boldsymbol{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

Homogeneous coordinates

Here is a point in 3D:

$$\boldsymbol{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

What if we made it more complicated and used four numbers?

$$\boldsymbol{P_h} = \begin{bmatrix} sP_x \\ sP_y \\ sP_z \\ s \end{bmatrix}$$

Uhm, okay?

So this means that

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \\ 6 \\ 2 \end{bmatrix}$$

are the same point in homogeneous coordinates

Euclidean transformations again

Rotation $\boldsymbol{R} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix}$, with columns $\boldsymbol{r}_i$, and translation $\boldsymbol{t}$

$$\boldsymbol{P}_1 = \boldsymbol{R}\boldsymbol{P}_0 + \boldsymbol{t}$$

Euclidean transformations again

Rotation $\boldsymbol{R} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix}$, with columns $\boldsymbol{r}_i$, and translation $\boldsymbol{t}$

$$\boldsymbol{P}_1 = \boldsymbol{R}\boldsymbol{P}_0 + \boldsymbol{t} = \boldsymbol{r}_1 P_x + \boldsymbol{r}_2 P_y + \boldsymbol{r}_3 P_z + \boldsymbol{t}$$

Euclidean transformations again

Rotation $\boldsymbol{R} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix}$, with columns $\boldsymbol{r}_i$, and translation $\boldsymbol{t}$

$$\boldsymbol{P}_1 = \boldsymbol{R}\boldsymbol{P}_0 + \boldsymbol{t} = \boldsymbol{r}_1 P_x + \boldsymbol{r}_2 P_y + \boldsymbol{r}_3 P_z + \boldsymbol{t}$$

$$\boldsymbol{P}_1 = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

Euclidean transformations again

Rotation $\boldsymbol{R} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix}$, with columns $\boldsymbol{r}_i$, and translation $\boldsymbol{t}$

$$\boldsymbol{P}_1 = \boldsymbol{R}\boldsymbol{P}_0 + \boldsymbol{t} = \boldsymbol{r}_1 P_x + \boldsymbol{r}_2 P_y + \boldsymbol{r}_3 P_z + \boldsymbol{t}$$

$$\boldsymbol{P}_1 = \underbrace{\begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 & \boldsymbol{t} \end{bmatrix}}_{\text{Transform: } \tilde{\boldsymbol{T}}} \underbrace{\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}}_{\text{Homogeneous: } \boldsymbol{P}_{0h}} = \tilde{\boldsymbol{T}}\boldsymbol{P}_{0h}$$

Homogeneous euclidean transformations

Fully homogeneous euclidean transformations become

$$\boldsymbol{P}_{1h} = \boldsymbol{T}\boldsymbol{P}_{0h}$$

$$\begin{bmatrix} \boldsymbol{P}_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{P}_0 \\ 1 \end{bmatrix}$$

Homogeneous euclidean transformations

The homogeneous transformation $\boldsymbol{T}$ takes on the 4×4 form

$$\boldsymbol{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$$

Robot arm and homogeneous transformations

$$\boldsymbol{Q}_{4h} = \boldsymbol{T}_4\boldsymbol{T}_3\boldsymbol{T}_2\boldsymbol{T}_1\boldsymbol{Q}_{0h}$$

Not just easier; It is computationally faster!

The *in*homogeneous p corresponds to the homogeneous p_h by

$$\boldsymbol{p}_h = \begin{bmatrix} s\boldsymbol{p} \\ s \end{bmatrix}$$

The projective transformation

Assume $f = 1$:

$$p_x = \frac{P_x}{P_z}, \quad p_y = \frac{P_y}{P_z}$$

$$\boldsymbol{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} sp_x \\ sp_y \\ s \end{bmatrix} = \boldsymbol{p}_h$$

Projective transformation is like assuming point in 3D is a 2D homogeneous point.

There is no standard notation for homogeneous coordinates

$$\boldsymbol{q} = \begin{bmatrix} s\boldsymbol{p} \\ s \end{bmatrix}$$

Lines in homogenous coordinates

Consider the line given implicitly by

$$0 = ax + by + c$$

We can write this in homogeneous coordinates

$$\begin{aligned} &= \begin{bmatrix} a \\ b \\ c \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} \\ &= \boldsymbol{l}^{\mathsf{T}} \boldsymbol{p}_h \end{aligned}$$

Lines in homogenous coordinates

If $a^2 + b^2 = 1$ and the scale of the homogeneous point is 1 then,

$$
\begin{aligned}
d &= \begin{bmatrix} a \\ b \\ c \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\
&= \boldsymbol{l}^{\mathsf{T}} \boldsymbol{p}_h
\end{aligned}
$$

d is the signed distance from the point to the line.

The homogeneous coordinate system — summary

The additional imaginary scale s, or alternatively dimension w

$$\boldsymbol{u} = s \begin{bmatrix} \boldsymbol{v} \\ 1 \end{bmatrix} = \begin{bmatrix} s\boldsymbol{v} \\ s \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}' \\ w \end{bmatrix}$$

- Dimensionality is $N + 1$
- Points have a scale $s \neq 0$
- Directions have $w = 0$
- Many-to-one correspondence: $\boldsymbol{u} \in \mathbb{R}^{N+1}$ and $\boldsymbol{v} \in \mathbb{R}^{N}$

Getting *in*homogeneous coordinates back

$$\boldsymbol{v} = \Pi\left(\boldsymbol{u}\right) = \Pi\left(\begin{bmatrix}\boldsymbol{v}' \\ s\end{bmatrix}\right) = \boldsymbol{v}'/s$$

Trivial inverse

Pinhole camera model

Pinhole camera

Image on your computer

Image from perspective projection

Where is (0, 0) in these images?

Principal point

- Image on your computer
  - Upper left corner
- Image from projective transformation
  - Optical centre (approximately in the middle of the image)

We introduce δ_x and δ_y to move these points to the same place.

This is called the principal point.

Principal point

Projection is now

$$p_x = \frac{f}{P_z} P_x + \delta_x$$
$$p_y = \frac{f}{P_z} P_y + \delta_y$$

Can we write all of this using homogeneous coordinates?

Principal point

Yes we can!

$$\boldsymbol{p}_h = \underbrace{\begin{bmatrix} f & 0 & \delta_x \\ 0 & f & \delta_y \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{K}} \boldsymbol{P}$$

And it even looks nice! This is called the camera matrix.

It contains intrinsic camera parameters.

Extrinsics

- The extrinsics of the camera are the rotation ($\boldsymbol{R}$) and translation ($\boldsymbol{t}$).
- They describe the pose of the camera.

Extrinsics

- The extrinsics of the camera are the rotation ($\boldsymbol{R}$) and translation ($\boldsymbol{t}$).
- They describe the pose of the camera.
- To project points, we first transform them to the reference frame of the camera:

$$\begin{aligned}\boldsymbol{P}_{cam} &= \boldsymbol{R}\boldsymbol{P} + \boldsymbol{t} \\ &= \begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\end{bmatrix}\begin{bmatrix}\boldsymbol{P} \\ 1\end{bmatrix}\end{aligned}$$

Projection matrix

Projecting a single point in 3D to the camera

$$\boldsymbol{p}_h = \boldsymbol{K}\boldsymbol{P}_{cam}$$

Projection matrix

Projecting a single point in 3D to the camera

$$
\begin{aligned}
\boldsymbol{p}_h &= \boldsymbol{K}\boldsymbol{P}_{cam} \\
&= \underbrace{\boldsymbol{K}\begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\end{bmatrix}}_{\mathscr{P}}\boldsymbol{P}_h
\end{aligned}
$$

Projection matrix

Projecting a single point in 3D to the camera

$$\begin{aligned}\boldsymbol{p}_h &= \boldsymbol{K}\boldsymbol{P}_{cam} \\ &= \underbrace{\boldsymbol{K}\begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\end{bmatrix}}_{\mathscr{P}}\boldsymbol{P}_h =\end{aligned}$$

$$\begin{bmatrix} sp_x \\ sp_y \\ s \end{bmatrix} = \begin{bmatrix} f & 0 & \delta_x \\ 0 & f & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

Wrapping up

- The translation is not the position of the camera
- The final coordinate of $\boldsymbol{P}_h$ must be 1.

Projection matrix:

$$\boldsymbol{q} = \boldsymbol{\mathcal{P}}\boldsymbol{P}_h = \boldsymbol{K}\begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\end{bmatrix}\boldsymbol{P}_h$$

The matrix $\mathcal{P}$ is known as the

projection matrix

(don't call it the camera matrix)

Exercise information

- Use Python interactively
  - Jupyter notebook
  - VS Code
  - Spyder
  - etc.
- Makes it easier to debug

Exercise information: Storing points on the computer

- Storing multiple one-dimensional vectors happens frequently
- Matrices are ideal for this
- We always operate on column vectors, so these matrices should be $3 \times n$ for a 3D vector (for example)
- Matrix multiplication lets you project many points at once
  - `ph = P.dot(Ph)` or even shorter
  - `ph = P@Ph`

Comment about exercises

- If you need for-loops, you're probably not doing it the easy way.
  - No exercises today need for-loops (except the provided function)
- Converting from homogeneous coordinates to regular
  - `p = ph[:-1]/ph[-1]`
- Ask the TAs 🙂

Learning objectives

After this lecture you should be able to:

- explain homogeneous coordinates
- convert to and from homogeneous coordinates
- perform relevant coordinate transformations
- explain the pinhole camera model

Exercise time!

Camera model and homographies

Camera parameters; lens distortion; homography estimation

Morten R. Hannemose, mohan@dtu.dk

February 10, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

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 ∞

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.

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

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

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

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

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]

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

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

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 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

H

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 $s\boldsymbol{q}_2$ 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 (next slide) ...

Homography rewritten ii

$$
\begin{aligned}
\mathbf{0} &= \boldsymbol{q}_{1i} \times \boldsymbol{H}\boldsymbol{q}_{2i}\,, \\
&= \boldsymbol{B}^{(i)}\mathsf{flatten}(\boldsymbol{H}^{\mathsf{T}})\,,
\end{aligned}
$$

$$
\boldsymbol{B}^{(i)} = \begin{bmatrix}
0 & -x_{2i} & x_{2i}y_{1i} & 0 & -y_{2i} & y_{2i}y_{1i} & 0 & -1 & y_{1i} \\
x_{2i} & 0 & -x_{2i}x_{1i} & y_{2i} & 0 & -y_{2i}x_{1i} & 1 & 0 & -x_{1i} \\
-x_{2i}y_{1i} & x_{2i}x_{1i} & 0 & -y_{2i}y_{1i} & y_{2i}x_{1i} & 0 & -y_{1i} & x_{1i} & 0
\end{bmatrix},
$$

$$
\mathsf{flatten}(\boldsymbol{H}^{\mathsf{T}}) = \begin{bmatrix} H_{11} & H_{21} & H_{31} & H_{12} & H_{22} & H_{32} & H_{13} & H_{23} & H_{33} \end{bmatrix}^{\mathsf{T}}
$$

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

The $\boldsymbol{B}^{(i)}$ matrices can be implemented with the Kronecker product:

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

Homography matrix solution

Given n corresponding points, we define $\boldsymbol{B}$

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

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

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

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

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

Homogeneous least-squares solution

Which eigenvector?

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

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

where e_i is the vector with 0 everywhere except for the ith 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`)

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

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

Least-squares solution with SVD

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

Stereo view geometry

Epipolar geometry, triangulation

Morten R. Hannemose, mohan@dtu.dk

February 17, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- Derive and explain the epipolar line in computer vision
- Derive and apply the fundamental matrix in computer vision
- Derive and apply the essential matrix in computer vision
- Implement the linear algorithm for triangulation
- Explain the pros and cons of using a linear algorithm

Presentation topics

- Projection of points, lines and planes
- Epipolar planes and lines
- Essential and fundamental matrices
- Triangulation
  - Problems with linear algorithm

Projection of points, lines and planes

Point projections

Any 3D point $\boldsymbol{P}$ projects to a 2D point $\boldsymbol{p}$:

$$\boldsymbol{P} \xrightarrow{\text{projection}} \boldsymbol{p}\,.$$

Line projections

Any 3D line $\boldsymbol{L}$ projects to a 2D line $\boldsymbol{l}$, except if $\boldsymbol{L} \parallel \boldsymbol{L}_{\mathscr{P}}$:

$$\boldsymbol{L}_0 \xrightarrow{\text{projection}} \boldsymbol{l}_0 \,,$$

$$\boldsymbol{L}_1 \xrightarrow{\text{projection}} \boldsymbol{p}_1 \,.$$

Plane projections

Any 3D plane $\boldsymbol{P}$ projects to the image plane,

$$\boldsymbol{P} \xrightarrow{\text{projection}} \boldsymbol{p}\,,$$

except if $\boldsymbol{P} \parallel \boldsymbol{L}_{\mathcal{P}}$

Where then?

Epipolar planes and lines

Epipolar lines

- Throwback to the demonstration from the first week
- A point seen in one camera must lie along a specific 3D line
- A point seen in one camera must lie along a specific 2D line in the other camera
  - This is the epipolar line!

Stereo view

Assume stereo view (two cameras)

Epipolar planes

The camera centers ($\boldsymbol{q}_1$ and $\boldsymbol{q}_2$) and the 3D point $\boldsymbol{Q}$ form an epipolar plane!

Epipolar planes

Which line in space is alway part of the epipolar planes?

Epipolar lines

The epipolar lines intersect the epipolar plane and images!

Epipolar lines

The relation between the line $q_i Q$ and the epipolar lines?

Epipolar lines

All epipolar lines intersect in the epipoles.

Epipoles

Where are the epipoles?

Stereo view is an epipolar geometry:
Each Q in 3D has an epipolar plane.
Pixels correspond if and only if
they lie in the same epipolar plane

Essential and fundamental matrices

Setting the scene

Let $\boldsymbol{q}$ refer to a 2D point in pixels and $\boldsymbol{p}$ refer to the same point in 3D in the reference frame of the camera.

These are related as follows:

$$\boldsymbol{q} = \boldsymbol{K}\boldsymbol{p}$$
$$\boldsymbol{p} = \boldsymbol{K}^{-1}\boldsymbol{q}$$

$\boldsymbol{R}$, $\boldsymbol{t}$ maps from the reference frame of camera one to the reference frame of camera two (their relative transformation)

The essential matrix

Consider the epipolar plane given by $\boldsymbol{p}_1$

The essential matrix

Relative to camera 2, what is the normal of the epipolar plane?

The normal is orthogonal to $\boldsymbol{t}$ and $\boldsymbol{R}\boldsymbol{p}_1 + \boldsymbol{t}$

The essential matrix

The normal is then

$$\begin{aligned} \boldsymbol{n} &= \boldsymbol{t} \times (\boldsymbol{R}\boldsymbol{p}_1 + \boldsymbol{t}) \\ &= \boldsymbol{t} \times (\boldsymbol{R}\boldsymbol{p}_1) \\ &= \underbrace{[\boldsymbol{t}]_\times \boldsymbol{R}}_{\boldsymbol{E}} \boldsymbol{p}_1. \end{aligned}$$

$\boldsymbol{E}$ is called the essential matrix!

The essential matrix

Dot product of orthogonal vectors are zero.

For the corresponding $\boldsymbol{p}_2$ in the second camera

$$\begin{aligned} 0 &= \boldsymbol{p}_2^\top \boldsymbol{n} \\ &= \boldsymbol{p}_2^\top \boldsymbol{E} \boldsymbol{p}_1 \end{aligned}$$

The essential matrix imposes a constraint on corresponding points.

How to interpret this?

- $\boldsymbol{p} = \boldsymbol{K}^{-1}\boldsymbol{q}$.
- Are $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ in homogeneous or inhomogeneous coordinates?

How to interpret this?

- $\boldsymbol{p} = \boldsymbol{K}^{-1}\boldsymbol{q}$.
- Are $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ in homogeneous or inhomogeneous coordinates?
  - Yes!...
- There are two interpretations:

How to interpret this?

- $\boldsymbol{p} = \boldsymbol{K}^{-1}\boldsymbol{q}$.
- Are $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ in homogeneous or inhomogeneous coordinates?
  - Yes!...
- There are two interpretations:
  - $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are 3D points and $\boldsymbol{n}$ is a vector in 3D. We use this in our derivations.
  - $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are 2D points and $\boldsymbol{n} = \boldsymbol{E}\boldsymbol{p}_1$ is the epipolar line, both are in homogeneous coordinates.
- 🤯

The fundamental matrix

Recall that

$$\boldsymbol{p} = \boldsymbol{K}^{-1}\boldsymbol{q}.$$

Then

$$\begin{aligned} \boldsymbol{p}_2^\top \boldsymbol{E} \boldsymbol{p}_1 &= 0 \\ (\boldsymbol{K}_2^{-1}\boldsymbol{q}_2)^\top \boldsymbol{E} (\boldsymbol{K}_1^{-1}\boldsymbol{q}_1) &= 0 \\ \boldsymbol{q}_2^\top \underbrace{\boldsymbol{K}_2^{-\top} \boldsymbol{E} \boldsymbol{K}_1^{-1}}_{\boldsymbol{F}} \boldsymbol{q}_1 &= 0 \end{aligned}$$

where $\boldsymbol{F}$ is the fundamental matrix!

The essential and fundamental matrices form requirements for pixel correspondence:

$$\boldsymbol{p}_2^\mathsf{T} \boldsymbol{E} \boldsymbol{p}_1 = 0$$
$$\boldsymbol{q}_2^\mathsf{T} \boldsymbol{F} \boldsymbol{q}_1 = 0$$

Note on $\boldsymbol{R}$, $\boldsymbol{t}$

What if $\boldsymbol{R}$, $\boldsymbol{t}$ is not given, but you only know the pose of each camera in world coordinates ($\boldsymbol{R}_1$, $\boldsymbol{t}_1$, $\boldsymbol{R}_2$, $\boldsymbol{t}_2$)?

You can compute the relative transformation (which you will do in the exercise today)

Fundamental/essential matrix vs homography

What is the difference?

Fundamental/essential matrix vs homography

What is the difference?

- The fundamental and essential matrices yield epipolar lines:
  - $0 = \boldsymbol{p}_2^\top \boldsymbol{E} \boldsymbol{p}_1$
  - $0 = \boldsymbol{q}_2^\top \boldsymbol{F} \boldsymbol{q}_1$
  - Corresponding points must lie on the epipolar line, no matter what is in the image.
- The homography establishes one-to-one correspondences:
  - $\boldsymbol{q}_2 = \boldsymbol{H} \boldsymbol{q}_1$
  - Only valid for planes.

Degrees of freedom of $\boldsymbol{F}$

- $\boldsymbol{F}$ has 9 numbers, and is scale invariant.
- $[\boldsymbol{t}]_\times$ has rank 2, and thus $\boldsymbol{F}$ has as well.
  - In other terms: $\det(\boldsymbol{F}) = 0$.
- You will estimate $\boldsymbol{F}$ in week 10.

Triangulation

Triangulation

We've seen the same point in many (known) cameras and want to find the point in 3D.

Because of noise, there is no exact solution.

Triangulation problem

Consider a projection matrix

$$\boldsymbol{\mathcal{P}}_i = \begin{bmatrix} \boldsymbol{p}_i^{(1)} \\ \boldsymbol{p}_i^{(2)} \\ \boldsymbol{p}_i^{(3)} \end{bmatrix}$$

where $\boldsymbol{p}_i^{(1)}$ is the ith row of $\boldsymbol{\mathcal{P}}_i$.

Triangulation equations

Projection gives the pixels

$$\boldsymbol{q}_i = \begin{bmatrix} s_i x_i \\ s_i y_i \\ s_i \end{bmatrix} = \boldsymbol{\mathcal{P}}_i \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{p}_i^{(1)} \boldsymbol{Q} \\ \boldsymbol{p}_i^{(2)} \boldsymbol{Q} \\ \boldsymbol{p}_i^{(3)} \boldsymbol{Q} \end{bmatrix}$$

This is two constraints $(x_i,\ y_i)$ in three equations.

Triangulation equations

As $s_i = \boldsymbol{p}_i^{(3)} \boldsymbol{Q}$, we have

$$\left(\boldsymbol{p}_i^{(3)} \boldsymbol{Q}\right) \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_i^{(1)} \boldsymbol{Q} \\ \boldsymbol{p}_i^{(2)} \boldsymbol{Q} \end{bmatrix}$$

rearranged into

$$\begin{aligned} \mathbf{0} &= \begin{bmatrix} \boldsymbol{p}_i^{(3)} x_i - \boldsymbol{p}_i^{(1)} \\ \boldsymbol{p}_i^{(3)} y_i - \boldsymbol{p}_i^{(2)} \end{bmatrix} \boldsymbol{Q} \\ &= \boldsymbol{B}^{(i)} \boldsymbol{Q} \end{aligned}$$

Triangulation solution

Define $\boldsymbol{B}$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}^{(1)} \\ \boldsymbol{B}^{(2)} \\ \vdots \\ \boldsymbol{B}^{(n)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_1^{(3)} x_1 - \boldsymbol{p}_1^{(1)} \\ \boldsymbol{p}_1^{(3)} y_1 - \boldsymbol{p}_1^{(2)} \\ \boldsymbol{p}_2^{(3)} x_2 - \boldsymbol{p}_2^{(1)} \\ \boldsymbol{p}_2^{(3)} y_2 - \boldsymbol{p}_2^{(2)} \\ \vdots \end{bmatrix}.$$

Use SVD to find $\arg\min_{\boldsymbol{Q}} \|\boldsymbol{B}\boldsymbol{Q}\|_2, \quad \text{s.t. } \|\boldsymbol{Q}\|_2 = 1.$

Linear algorithm, hmm

- $\arg\min_{\boldsymbol{Q}} \|\boldsymbol{B}\boldsymbol{Q}\|_2 \,, \quad$ s.t. $\|\boldsymbol{Q}\|_2 = 1.$
- This is a linear algorithm used to solve the problem.
- Which error would we actually like to minimize?

Linear algorithm, hmm

- $\arg\min_{\boldsymbol{Q}} \|\boldsymbol{B}\boldsymbol{Q}\|_2 \,, \quad \text{s.t. } \|\boldsymbol{Q}\|_2 = 1.$
- This is a linear algorithm used to solve the problem.
- Which error would we actually like to minimize?
  - Depends why we believe that there are errors
  - Errors in the observed pixel location

Triangulation: ideal error

- Let $\begin{bmatrix}\tilde{x}_i & \tilde{y}_i\end{bmatrix}$ and $\begin{bmatrix}x_i & y_i\end{bmatrix}$ refer to the observed and projected pixel coordinates, respectively.
- Which error would we like to minimize?

$$e_{\text{ideal}} = \sum_i \left\| \begin{bmatrix}\tilde{x}_i \\ \tilde{y}_i\end{bmatrix} - \begin{bmatrix}x_i \\ y_i\end{bmatrix} \right\|_2^2$$

Triangulation uh oh!

- Which error are actually we minimizing?

$$
\begin{aligned}
e_{\text{algebraic}} &= \sum_i \|\boldsymbol{B}_i \boldsymbol{Q}\|_2^2 \\
&= \sum_i \left\| \begin{bmatrix} \boldsymbol{p}_i^{(3)} \tilde{x}_i - \boldsymbol{p}_i^{(1)} \\ \boldsymbol{p}_i^{(3)} \tilde{x}_i - \boldsymbol{p}_i^{(2)} \end{bmatrix} \boldsymbol{Q} \right\|_2^2 \\
&= \sum_i \left\| \underbrace{\boldsymbol{p}_i^{(3)} \boldsymbol{Q}}_{s_i} \begin{bmatrix} \tilde{x}_i \\ \tilde{x}_i \end{bmatrix} - \begin{bmatrix} \boldsymbol{p}_i^{(1)} \\ \boldsymbol{p}_i^{(2)} \end{bmatrix} \boldsymbol{Q} \right\|_2^2 \\
&= \sum_i \left\| s_i \begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \end{bmatrix} - s_i \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right\|_2^2
\end{aligned}
$$

Error terms compared

- We can rewrite the terms slightly again

$$e_{\text{ideal}} = \sum_i \quad (\tilde{x}_i - x_i)^2 + \quad (\tilde{y}_i - y_i)^2$$

$$e_{\text{algebraic}} = \sum_i s_i^2 (\tilde{x}_i - x_i)^2 + s_i^2 (\tilde{y}_i - y_i)^2$$

- s_i is larger for cameras that are further from $\boldsymbol{Q}$

Linear vs non-linear algorithms

- e_{ideal} can be minimized using non-linear optimization
- Linear algorithms are very fast.
  - Only minimizes an algebraic error.
- We can estimate many things using linear algorithms
  - Triangulation, homography, fundamental matrix, projection matrix
  - They all have the problem that they don't minimize the exact error we desire
- Linear algorithms are acceptable in most cases.
- When high accuracy is desired initialize the non-linear optimization with the linear solution.

Learning objectives

After this lecture you should be able to:

- Derive and explain the epipolar line in computer vision
- Derive and apply the fundamental matrix in computer vision
- Derive and apply the essential matrix in computer vision
- Implement the linear algorithm for triangulation
- Explain the pros and cons of using a linear algorithm

Exercise time!

Camera calibration

Morten R. Hannemose, mohan@dtu.dk

February 24, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- implement and use the direct linear transformation to calibrate a pinhole camera
- implement and use Zhang's method (2000) to calibrate a pinhole camera using checkerboards

Presentation topics

- Direct linear transformation
- Zhang's method (2000)
  - Reprojection error
- Non-linear calibration
- Practical remarks

Culmination of previous weeks

- Pinhole camera model
- Homogeneous coordinates
- Homographies
- Linear algorithms
- Calibration

Direct linear transformation

Direct linear transformation

Start from the projection equation

$$\boldsymbol{q}_i = \boldsymbol{\mathcal{P}}\boldsymbol{Q}_i$$

$$\begin{bmatrix} sx_i \\ sy_i \\ s \end{bmatrix} = \boldsymbol{\mathcal{P}} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

then rearrange into the form $\boldsymbol{B}^{(i)}\text{flatten}(\boldsymbol{\mathcal{P}}^\top) = \boldsymbol{0}$.

I use the form explained in LN: 2.7 Camera Resection.

Direct linear transformation i

$$
\begin{aligned}
\boldsymbol{q}_i &= \boldsymbol{\mathcal{P}}\boldsymbol{Q}_i, \\
\mathbf{0} &= \boldsymbol{q}_i \times \boldsymbol{\mathcal{P}}\boldsymbol{Q}_i, \\
&= [\boldsymbol{q}_i]_\times \boldsymbol{\mathcal{P}}\boldsymbol{Q}_i, \\
&= \boldsymbol{B}^{(i)}\mathsf{flatten}(\boldsymbol{\mathcal{P}}^\mathsf{T}),
\end{aligned}
$$

where (continue to next slide) ...

Direct linear transformation ii

$$\mathbf{0} = \boldsymbol{B}^{(i)}\mathsf{flatten}(\boldsymbol{\mathcal{P}}^{\mathsf{T}}),$$

$$\boldsymbol{B}^{(i)} = \begin{bmatrix} 0 & -X_i & X_i y_i & 0 & -Y_i & Y_i y_i & 0 & -Z_i & Z_i y_i & 0 & -1 & y_i \\ X_i & 0 & -X_i x_i & Y_i & 0 & -Y_i x_i & Z_i & 0 & -Z_i x_i & 1 & 0 & -x_i \\ -X_i y_i & X_i x_i & 0 & -Y_i y_i & Y_i x_i & 0 & -Z_i y_i & Z_i x_i & 0 & -y_i & x_i & 0 \end{bmatrix},$$

$$= \boldsymbol{Q}_i \otimes [\boldsymbol{q}_i/s]_\times$$

$$\mathsf{flatten}(\boldsymbol{\mathcal{P}}^{\mathsf{T}}) = \begin{bmatrix} \mathcal{P}_{11} & \mathcal{P}_{21} & \mathcal{P}_{31} & \mathcal{P}_{12} & \mathcal{P}_{22} & \mathcal{P}_{32} & \mathcal{P}_{13} & \mathcal{P}_{23} & \mathcal{P}_{33} & \mathcal{P}_{14} & \mathcal{P}_{24} & \mathcal{P}_{34} \end{bmatrix}^{\mathsf{T}}$$

Direct linear transformation

Now let

$$\mathbf{0} = \boldsymbol{B}\ \mathsf{flatten}(\boldsymbol{\mathcal{P}}^{\mathsf{T}}),$$

where

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

and solve using SVD on $\boldsymbol{B}$.

Zhang's method (2000)

Using a checkerboard

www.calib.io | 8x11 | Checker Size: 15 mm.

Using a checkerboard

View the checkerboard in different poses:

Using a checkerboard

Problem:

Each view i has a different rotation $\boldsymbol{R}_i$ and translation $\boldsymbol{t}_i$

How do we find all $\boldsymbol{\mathcal{P}}_i$?

Zhang's method

First, assume all checkerboard corners are in the $Z = 0$ plane:

$$\boldsymbol{Q}_j = \begin{bmatrix} X_j \\ Y_j \\ 0 \\ 1 \end{bmatrix}$$

Simplifying the projection equation

Let $\boldsymbol{r}_i^{(c)}$ is the cth column of $\boldsymbol{R}_i$. Now projection is

$$\boldsymbol{q}_{ij} = \boldsymbol{\mathcal{P}}_i \boldsymbol{Q}_j = \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_i^{(1)} & \boldsymbol{r}_i^{(2)} & \boldsymbol{r}_i^{(3)} & \boldsymbol{t}_i \end{bmatrix} \underbrace{\begin{bmatrix} X_j \\ Y_j \\ 0 \\ 1 \end{bmatrix}}_{\boldsymbol{Q}}$$

Simplifying the projection equation

Let $\boldsymbol{r}_i^{(c)}$ is the cth column of $\boldsymbol{R}_i$. Now projection is

$$\boldsymbol{q}_{ij} = \boldsymbol{\mathcal{P}}_i \boldsymbol{Q}_j = \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_i^{(1)} & \boldsymbol{r}_i^{(2)} & \boldsymbol{r}_i^{(3)} & \boldsymbol{t}_i \end{bmatrix} \underbrace{\begin{bmatrix} X_j \\ Y_j \\ 0 \\ 1 \end{bmatrix}}_{\boldsymbol{Q}}$$

$$= \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_i^{(1)} & \boldsymbol{r}_i^{(2)} & \boldsymbol{t}_i \end{bmatrix} \underbrace{\begin{bmatrix} X_j \\ Y_j \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{Q}}_j}.$$

From projections to homographies

$$\boldsymbol{q}_{ij} = \underbrace{\boldsymbol{K}\begin{bmatrix}\boldsymbol{r}_i^{(1)} & \boldsymbol{r}_i^{(2)} & \boldsymbol{t}_i\end{bmatrix}}_{\boldsymbol{H}_i}\underbrace{\begin{bmatrix}X_j\\Y_j\\1\end{bmatrix}}_{\tilde{\boldsymbol{Q}}_j}$$

$$= \boldsymbol{H}_i\tilde{\boldsymbol{Q}}_j$$

The homographies $\boldsymbol{H}_i$ can be determined from the plane-plane correspondence $\tilde{\boldsymbol{Q}}_j$ to $\boldsymbol{q}_{ij}$ (week 2).

Corner correspondences

Need to find the unique positions of corners $\boldsymbol{q}_{ij}$.

Find all H_i from corners $\tilde{Q}_j$ and projections q_{ij} with $q_{ij} = H_i \tilde{Q}_j$

For example, using SVD.

From homographies to the camera matrix

$$\boldsymbol{H}_i = \begin{bmatrix} \boldsymbol{h}_i^{(1)} & \boldsymbol{h}_i^{(2)} & \boldsymbol{h}_i^{(3)} \end{bmatrix} = \lambda_i \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_i^{(1)} & \boldsymbol{r}_i^{(2)} & \boldsymbol{t}_i \end{bmatrix}.$$

$\boldsymbol{r}_i^{(1)}$ and $\boldsymbol{r}_i^{(2)}$ are orthonormal, i.e.

$$\begin{aligned} \boldsymbol{r}_i^{(1)\mathsf{T}} \boldsymbol{r}_i^{(1)} &= \boldsymbol{r}_i^{(2)\mathsf{T}} \boldsymbol{r}_i^{(2)} = 1, \\ \boldsymbol{r}_i^{(1)\mathsf{T}} \boldsymbol{r}_i^{(2)} &= \boldsymbol{r}_i^{(2)\mathsf{T}} \boldsymbol{r}_i^{(1)} = 0. \end{aligned}$$

From homographies to the camera matrix

Express $\boldsymbol{r}_i^{(\alpha)}$ using $\boldsymbol{h}_i^{(\alpha)}$:

$$\boldsymbol{h}_i^{(\alpha)} = \lambda_i \boldsymbol{K} \boldsymbol{r}_i^{(\alpha)}, \Leftrightarrow$$

$$\boldsymbol{K}^{-1} \boldsymbol{h}_i^{(\alpha)} = \lambda_i \boldsymbol{r}_i^{(\alpha)}.$$

From homographies to the camera matrix

Express $\boldsymbol{r}_i^{(\alpha)}$ using $\boldsymbol{h}_i^{(\alpha)}$:

$$\boldsymbol{h}_i^{(\alpha)} = \lambda_i \boldsymbol{K} \boldsymbol{r}_i^{(\alpha)}, \Leftrightarrow$$
$$\boldsymbol{K}^{-1} \boldsymbol{h}_i^{(\alpha)} = \lambda_i \boldsymbol{r}_i^{(\alpha)}.$$

Now the constraints from the previous slide are:

$$\boldsymbol{h}_i^{(1)\mathsf{T}} \boldsymbol{K}^{-\mathsf{T}} \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(2)} = 0,$$
$$\boldsymbol{h}_i^{(1)\mathsf{T}} \boldsymbol{K}^{-\mathsf{T}} \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(1)} = \boldsymbol{h}_i^{(2)\mathsf{T}} \boldsymbol{K}^{-\mathsf{T}} \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(2)} = \lambda_i^2.$$

We have found constraints on the camera matrix! 🥳

Number of constraints

- Two constraints doesn't seem that impressive?

Number of constraints

- Two constraints doesn't seem that impressive?
- Homography has eight degrees of freedom
- Pose of checkerboard has six (3 rotation, 3 translation)
- A single homography can only fix two degrees of freedom of a camera matrix.

Define some new variables i

How to put into practice?

Define the matrix:

$$\boldsymbol{B} = \boldsymbol{K}^{-\mathsf{T}}\boldsymbol{K}^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix},$$

$$\boldsymbol{b} = \begin{bmatrix} B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33} \end{bmatrix}^{\mathsf{T}},$$

Define some new variables ii

Now define $\boldsymbol{v}_i^{(\alpha\beta)}$ such that

$$\boldsymbol{v}_i^{(\alpha\beta)}\boldsymbol{b} = \boldsymbol{h}_i^{(\alpha)^T}\boldsymbol{B}\,\boldsymbol{h}_i^{(\beta)}.$$

Then $\boldsymbol{v}_i^{(\alpha\beta)}$ must be an 1×6 vector given by

$$\begin{aligned}\boldsymbol{v}_i^{(\alpha\beta)} = [&H_i^{(1\alpha)}H_i^{(1\beta)}, \quad H_i^{(1\alpha)}H_i^{(2\beta)} + H_i^{(2\alpha}H_i^{(1\beta)}, \quad H_i^{(2\alpha)}H_i^{(2\beta)}, \quad \ldots \\ &H_i^{(3\alpha)}H_i^{(1\beta)} + H_i^{(1\alpha)}H_i^{(3\beta)}, \quad H_i^{(3\alpha)}H_i^{(2\beta)} + H_i^{(2\alpha)}H_i^{(3\beta)}, \quad H_i^{(3\alpha)}H_i^{(3\beta)}],\end{aligned}$$

Define some new variables iii

where $H_i^{(rc)}$ is the element in row r and column c of $\boldsymbol{H}_i$.

From homographies to the camera matrix

Recall our constraints:

$$\boldsymbol{h}_i^{(1)\mathsf{T}} \boldsymbol{K}^{-\mathsf{T}} \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(1)} = \boldsymbol{h}_i^{(2)\mathsf{T}} \boldsymbol{K}^{-T} \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(2)} \Leftrightarrow$$

$$\boldsymbol{h}_i^{(1)\mathsf{T}} \boldsymbol{B}\, \boldsymbol{h}_i^{(1)} - \boldsymbol{h}_i^{(2)\mathsf{T}} \boldsymbol{B}\, \boldsymbol{h}_i^{(2)} = 0 =$$

$$\left(\boldsymbol{v}_i^{(11)} - \boldsymbol{v}_i^{(22)}\right) \boldsymbol{b} = 0.$$

Now we can express our constraints in matrix-form

$$\begin{bmatrix} \boldsymbol{v}_i^{(12)} \\ \boldsymbol{v}_i^{(11)} - \boldsymbol{v}_i^{(22)} \end{bmatrix} \boldsymbol{b} = \boldsymbol{0}$$

From homographies to the camera matrix

For all checkerboard poses

$$\boldsymbol{V}\boldsymbol{b} = \begin{bmatrix} \boldsymbol{v}_1^{(12)} \\ \boldsymbol{v}_1^{(11)} - \boldsymbol{v}_1^{(22)} \\ \boldsymbol{v}_2^{(12)} \\ \boldsymbol{v}_2^{(11)} - \boldsymbol{v}_2^{(22)} \\ \vdots \end{bmatrix} \boldsymbol{b} = \boldsymbol{0}$$

When $\boldsymbol{b}$ is found then $\boldsymbol{K}$ can be determined using the formulas in Zhang's paper.

Find the camera matrix K through b using $Vb = 0$ and SVD, where V is built from the homographies H_i.

Reprojection error

Is it a good camera calibration? How to find out?

Reprojection error

Is it a good camera calibration? How to find out?

Reproject the points from the checkerboards to the camera, and compare to where we've seen them

$$\tilde{\boldsymbol{q}}_{ij} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R}_i & \boldsymbol{t}_i \end{bmatrix} \boldsymbol{Q}_j$$

$$\Pi(\tilde{\boldsymbol{q}}_{ij}) - \Pi(\boldsymbol{q}_{ij})$$

Reprojection error

We can now compute the reprojection error as the root mean squared error (RMSE)

$$\sqrt{\frac{1}{n}\sum_i\sum_j \left\|\Pi(\tilde{\boldsymbol{q}}_{ij}) - \Pi(\boldsymbol{q}_{ij})\right\|_2^2}$$

where n is the total number of points.
We have $\boldsymbol{K}$, but we are still missing something.

Reprojection error

We can now compute the reprojection error as the root mean squared error (RMSE)

$$\sqrt{\frac{1}{n}\sum_i \sum_j \left\| \Pi(\tilde{\boldsymbol{q}}_{ij}) - \Pi(\boldsymbol{q}_{ij}) \right\|_2^2}$$

where n is the total number of points.

We have $\boldsymbol{K}$, but we are still missing something.

How do we recover $\boldsymbol{R}_i$ and $\boldsymbol{t}_i$?

From homographies to poses

Recall: $\boldsymbol{H}_i = \begin{bmatrix} \boldsymbol{h}_i^{(1)} & \boldsymbol{h}_i^{(2)} & \boldsymbol{h}_i^{(3)} \end{bmatrix} = \lambda_i \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_i^{(1)} & \boldsymbol{r}_i^{(2)} & \boldsymbol{t}_i \end{bmatrix}$

Now we can recover $\boldsymbol{R}_i$ and $\boldsymbol{t}_i$:

$$\begin{aligned} \boldsymbol{r}_i^{(1)} &= \tfrac{1}{\lambda_i} \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(1)}, \\ \boldsymbol{r}_i^{(2)} &= \tfrac{1}{\lambda_i} \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(2)}, \\ \boldsymbol{t}_i &= \tfrac{1}{\lambda_i} \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(3)}, \\ \boldsymbol{r}_i^{(3)} &= \boldsymbol{r}_i^{(1)} \times \boldsymbol{r}_i^{(2)}, \end{aligned}$$

where $\lambda_i = \left\| \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(1)} \right\|_2 = \left\| \boldsymbol{K}^{-1} \boldsymbol{h}_i^{(2)} \right\|_2$.

Homography and poses

$$\boldsymbol{t}_i = \frac{1}{\lambda_i}\boldsymbol{K}^{-1}\boldsymbol{h}_i^{(3)}$$

What happens if $t_{iz} < 0$? And what does that mean?

Can we ensure correctness?

Pinhole camera revisited

Points behind the camera also get projected to the image plane.
Multiply all 3D points by -1 and they still project to the same place.

Homography and poses

- If $t_{iz} < 0$ the checkerboard is behind the camera.
- How to get the correct solution?
- Estimate $\boldsymbol{R}_i$ and $\boldsymbol{t}_i$ again using $-\boldsymbol{H}_i$ (flipped sign).

Non-linear calibration

Least-squares method

With projection equation

$$\boldsymbol{q}_{ij} = \boldsymbol{K}\left[\boldsymbol{R}_i \quad \boldsymbol{t}_i\right]\boldsymbol{Q}_j$$

then

$$E = \sum_{i,j}\left\|\mathsf{dist}\Big(\pi\left(\mathbf{K}\left[\mathbf{R}_i \quad \mathbf{t}_i\right]\boldsymbol{Q}_j\right)\Big) - \pi(\boldsymbol{q}_{ij})\right\|^2$$

Solution: minimize E w.r.t. $\mathbf{K}$, $\mathbf{R}_i$, $\mathbf{t}_i$ and lens distortion.

Practical remarks

What images should we take?

- Our two constraints per image are based on $\boldsymbol{r}_i^{(1)}$ and $\boldsymbol{r}_i^{(2)}$.
- Two parallel checkerboards express the same constraints.

What images should we take?

- Our two constraints per image are based on $\boldsymbol{r}_i^{(1)}$ and $\boldsymbol{r}_i^{(2)}$.
- Two parallel checkerboards express the same constraints.
  - at least when we don't consider lens distortion
- Make sure to rotate the checkerboards so they are not parallel.
- Make the checkerboards take up as as much of the frame as possible!

How many images?

- Without lens distortion: In theory at least three.
- In practice: It depends...
  - Some people use 2000+ images for a single calibration[1]
  - Look at the reprojection error
  - Both of the training set and the validation set

Learning objectives

After this lecture you should be able to:

- implement and use the direct linear transformation to calibrate a pinhole camera
- implement and use Zhang's method (2000) to calibrate a pinhole camera using checkerboards

Last week's exercise

The goal of exercise 3.6-3.10 last week was to:

- Understand how to handle when no camera has $\boldsymbol{R} = \boldsymbol{I}$ and $\boldsymbol{t} = \boldsymbol{0}$.
- Easily find epipolar lines in image 1 from points in image 2.

I have uploaded a new `TwoImageDataCar.npy`, where both have nontrivial $\boldsymbol{R}$ and $\boldsymbol{t}$.
For exercise 3.10, you can use the transpose.

$$\boldsymbol{q}_1^\top \boldsymbol{F} \boldsymbol{q}_2 = 0$$
$$\boldsymbol{q}_2^\top \boldsymbol{F}^\top \boldsymbol{q}_1 = 0$$

If you have already solved the exercise, loading the new file should not take long

Exercise time!

Nonlinear optimization

camera calibration and triangulation

Morten R. Hannemose, mohan@dtu.dk

March 3, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- perform non-linear optimization within computer vision
- explain the principle behind Levenberg-Marquardt
- compute the Jacobian of a function
- reason about different parameterizations of rotations in optimization

Presentation topics

- Non-linear least-squares optimization
  - Levenberg–Marquardt
- Gradients
  - Analytical gradients
  - Finite differences approximation
- Rotations in optimization
- Camera calibration

Non-linear least-squares optimization

Least-squares problems

Computer vision has many problems of the form:

$$\min_{\boldsymbol{x}} \|g(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 \,.$$

Make all parameters into a vector $\boldsymbol{x}$ and optimize everything!

Least-squares problems

Computer vision has many problems of the form:

$$\min_{\boldsymbol{x}} \|g(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 .$$

Make all parameters into a vector $\boldsymbol{x}$ and optimize everything!

- Homography estimation
- Pose estimation
- Bundle adjustment
- Camera calibration with and without lens distortion
- Triangulation
- ...

Reformulate a bit

$$
\begin{aligned}
e(\boldsymbol{x}) &= || \underbrace{g(\boldsymbol{x}) - \boldsymbol{y}}_{f(\boldsymbol{x})} ||_2^2 \\
&= \|f(\boldsymbol{x})\|_2^2 \\
&= f(\boldsymbol{x})^\top f(\boldsymbol{x})
\end{aligned}
$$

Levenberg-Marquardt

We solve the problem in an iterative fashion. At the kth iteration:

Replace with f with first order approximation around $\boldsymbol{x}_k$

$$f(\boldsymbol{x}_k + \boldsymbol{\delta}) \approx f(\boldsymbol{x}_k) + \boldsymbol{J}\boldsymbol{\delta}$$

$\boldsymbol{J}$ is the Jacobian that contains all first order derivatives of f at $\boldsymbol{x}_k$.

Levenberg-Marquardt

We solve the problem in an iterative fashion. At the kth iteration:

Replace with f with first order approximation around $\boldsymbol{x}_k$

$$f(\boldsymbol{x}_k + \boldsymbol{\delta}) \approx f(\boldsymbol{x}_k) + \boldsymbol{J}\boldsymbol{\delta}$$

$\boldsymbol{J}$ is the Jacobian that contains all first order derivatives of f at $\boldsymbol{x}_k$.

How are the the Jacobians of f and g related?

Levenberg-Marquardt

We solve the problem in an iterative fashion. At the kth iteration:

Replace with f with first order approximation around $\boldsymbol{x}_k$

$$f(\boldsymbol{x}_k + \boldsymbol{\delta}) \approx f(\boldsymbol{x}_k) + \boldsymbol{J}\boldsymbol{\delta}$$

$\boldsymbol{J}$ is the Jacobian that contains all first order derivatives of f at $\boldsymbol{x}_k$.

How are the the Jacobians of f and g related?
They are identical!

Levenberg-Marquardt

The sum of squared errors is then:

$$
\begin{aligned}
e(\boldsymbol{x}_k + \boldsymbol{\delta}) &= \|f(\boldsymbol{x}_k + \boldsymbol{\delta})\|_2^2 \approx \\
\tilde{e}(\boldsymbol{x}_k + \boldsymbol{\delta}) &= \big(f(\boldsymbol{x}_k) + \boldsymbol{J}\boldsymbol{\delta}\big)^\mathsf{T} \big(f(\boldsymbol{x}_k) + \boldsymbol{J}\boldsymbol{\delta}\big)
\end{aligned}
$$

Levenberg-Marquardt

The sum of squared errors is then:

$$
\begin{aligned}
e(\boldsymbol{x}_k + \boldsymbol{\delta}) &= \|f(\boldsymbol{x}_k + \boldsymbol{\delta})\|_2^2 \approx \\
\tilde{e}(\boldsymbol{x}_k + \boldsymbol{\delta}) &= \big(f(\boldsymbol{x}_k) + \boldsymbol{J}\boldsymbol{\delta}\big)^\mathsf{T}\big(f(\boldsymbol{x}_k) + \boldsymbol{J}\boldsymbol{\delta}\big) \\
&= f(\boldsymbol{x}_k)^\mathsf{T} f(\boldsymbol{x}_k) + 2(\boldsymbol{J}\boldsymbol{\delta})^\mathsf{T} f(\boldsymbol{x}_k) + (\boldsymbol{J}\boldsymbol{\delta})^\mathsf{T}\boldsymbol{J}\boldsymbol{\delta}^\mathsf{T} \\
&= f(\boldsymbol{x}_k)^\mathsf{T} f(\boldsymbol{x}_k) + 2\boldsymbol{\delta}^\mathsf{T}\boldsymbol{J}^\mathsf{T} f(\boldsymbol{x}_k) + \boldsymbol{\delta}^\mathsf{T}\boldsymbol{J}^\mathsf{T}\boldsymbol{J}\boldsymbol{\delta}^\mathsf{T}
\end{aligned}
$$

which is a second order approximation of e using only first order derivatives of f 😀

Levenberg-Marquardt

We can minimize our approximation instead.

Take the derivative of $\tilde{e}(\boldsymbol{x}_k + \boldsymbol{\delta})$

$$\tilde{e}(\boldsymbol{x}_k + \boldsymbol{\delta}) = f(\boldsymbol{x}_k)^\mathsf{T} f(\boldsymbol{x}_k) + 2\boldsymbol{\delta}^\mathsf{T}\boldsymbol{J}^\mathsf{T} f(\boldsymbol{x}_k) + \boldsymbol{\delta}^\mathsf{T}\boldsymbol{J}^\mathsf{T}\boldsymbol{J}\boldsymbol{\delta}^\mathsf{T}$$

$$\frac{\partial \tilde{e}(\boldsymbol{x}_k + \boldsymbol{\delta})}{\partial \boldsymbol{\delta}} = 2\boldsymbol{J}^\mathsf{T} f(\boldsymbol{x}_k) + 2\boldsymbol{J}^\mathsf{T}\boldsymbol{J}\boldsymbol{\delta}.$$

Levenberg-Marquardt

Find the optimum by setting the derivative equal to zero

$$2\boldsymbol{J}^\top f(\boldsymbol{x}_k) + 2\boldsymbol{J}^\top \boldsymbol{J}\boldsymbol{\delta} = \boldsymbol{0} \Leftrightarrow$$
$$\boldsymbol{J}^\top \boldsymbol{J}\boldsymbol{\delta} = -\boldsymbol{J}^\top f(\boldsymbol{x}_k).$$

Solve for $\boldsymbol{\delta}$, and then set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{\delta}$.

Rinse and repeat!

The λ parameter

The second order approximation of e is better the closer we are to the minimum.

When far away, it can be less good. What can we do in this case?

The λ parameter

The second order approximation of e is better the closer we are to the minimum.

When far away, it can be less good. What can we do in this case? Fall back to gradient descent!

$$(\boldsymbol{J}^\top \boldsymbol{J} + \lambda \boldsymbol{I})\boldsymbol{\delta} = -\boldsymbol{J}^\top f(\boldsymbol{x}_k).$$

Decrease λ when $e(\boldsymbol{x}_{k+1}) < e(\boldsymbol{x}_k)$, otherwise increase it.

Levenberg-Marquardt summary

Levenberg-Marquardt is often well suited to the types of problems we encounter in computer vision.

It exploits the nature of the least-squares problem to get a second order method, using only first order derivatives.

Gradients

Gradients

How do we compute $\boldsymbol{J}$?

Gradients

How do we compute $\boldsymbol{J}$?

Recall that $\boldsymbol{J}$ is just a lot of derivatives in a matrix.

- Analytical gradients
- Automatic differentiation
  - Reverse mode automatic differentiation (backpropagation)
  - Forward mode automatic differentiation (dual numbers)
- Finite differences approximation

Gradients

How do we compute $\boldsymbol{J}$?

Recall that $\boldsymbol{J}$ is just a lot of derivatives in a matrix.

- Analytical gradients
- Automatic differentiation
  - Reverse mode automatic differentiation (backpropagation)
  - Forward mode automatic differentiation (dual numbers)
- Finite differences approximation

Analytical gradients - how to find them?

Analytical gradients - how to find them?

Get out pen and paper (or Maple) and find the exact derivative.

Analytical gradients - triangulation example

The reprojection error of a point in 3D in multiple cameras

$$f(\boldsymbol{Q}) = \begin{bmatrix} \Pi\big(\boldsymbol{\mathcal{P}}_1 \Pi^{-1}(\boldsymbol{Q})\big) - \tilde{\boldsymbol{q}}_1 \\ \vdots \\ \Pi\big(\boldsymbol{\mathcal{P}}_n \Pi^{-1}(\boldsymbol{Q})\big) - \tilde{\boldsymbol{q}}_n \end{bmatrix}$$

- f returns a vector of length $2n$
- $\boldsymbol{Q}$ has three parameters
- thus $\boldsymbol{J}$ is $2n \times 3$.

Analytical gradients - triangulation example

The projected point in homogeneous coordinates:

$$\boldsymbol{q} = \boldsymbol{\mathcal{P}} \underbrace{\Pi^{-1}(\boldsymbol{Q})}_{\boldsymbol{Q}_h}$$

$$= \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}^{(1)} \\ \boldsymbol{p}^{(2)} \\ \boldsymbol{p}^{(3)} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Analytical gradients - triangulation example

Let's focus on x

$$x = \frac{\boldsymbol{p}^{(1)}\boldsymbol{Q}_h}{\boldsymbol{p}^{(3)}\boldsymbol{Q}_h} = \frac{\mathscr{P}_{11}X + \mathscr{P}_{12}Y + \mathscr{P}_{13}Z + \mathscr{P}_{14}}{\mathscr{P}_{31}X + \mathscr{P}_{32}Y + \mathscr{P}_{33}Z + \mathscr{P}_{34}}$$

A single element of $\boldsymbol{J}$ is then given by:

$$\frac{d}{dX}x = \frac{\mathscr{P}_{11}}{\boldsymbol{p}^{(3)}\boldsymbol{Q}_h} - \frac{\mathscr{P}_{31}(\boldsymbol{p}^{(1)}\boldsymbol{Q}_h)}{(\boldsymbol{p}^{(3)}\boldsymbol{Q}_h)^2}$$

Analytical gradients - summary

Analytical gradients are extremely useful

- + very fast to compute
- + accurate
- − complicated to derive and implement

Many functions in OpenCV return the Jacobian in addition the values themselves.

Finite differences approximation - forward differences

A first order Taylor expansion of f

$$f(x+h) = f(x) + \frac{d}{dx} f(x) h + O(h)$$

Finite differences approximation - forward differences

A first order Taylor expansion of f

$$f(x+h) = f(x) + \frac{d}{dx}f(x)h + O(h) \Leftrightarrow$$

can be rewritten to

$$\frac{d}{dx}f(x) = \frac{f(x+h) - f(x)}{h} + O(h).$$

This is called forward differences. (2-point)

Finite differences approximation - central differences

A second order Taylor expansion of f evaluated at $x - h$ and $x + h$ can be rearranged to yield

$$\frac{d}{dx} f(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).$$

This is called central differences, which is more accurate ($O(h^2)$), but requires two new evaluations of f.

Finite differences approximation - in practice

Principle can only be applied to one parameter at a time.

To compute $\boldsymbol{J}$ you need to evaluate f once (or twice) for each element in $\boldsymbol{x}$ (which can be a lot)

Number of evaluations can be reduced if you know the sparsity structure of $\boldsymbol{J}$, i.e. which elements of $\boldsymbol{x}$ affect which elements of $f(\boldsymbol{x})$

Finite differences approximation - h

How do we choose h?

A good choice is a fixed percentage of x.

Can this have any issues?

Finite differences approximation - h

How do we choose h?

A good choice is a fixed percentage of x.

Can this have any issues?

When x is zero, a percentage is also zero, this needs to be handled.

Finite differences approximation - summary

They only give an approximation of the gradients

- $+$ convenient
- $-$ lots of (unnecessary) computation
- $-$ has parameter h
- $-$ numeric problems

Only use when speed and robustness are not your primary concerns.

Rotations in optimization

Rotations in optimization

- Rotations are usually 3×3 matrices, but have just three degrees of freedom.
- How can we parametrize rotations?

Rotations in optimization

- Rotations are usually 3×3 matrices, but have just three degrees of freedom.
- How can we parametrize rotations?
  - optimizing all 9 numbers gives something that is no longer a rotation matrix

Rotations in optimization - Euler angles

Euler angles $(\theta_x, \theta_y, \theta_z)$ use only three numbers

- Suffers from gimbal lock, i.e. that two parameters do the same in certain configurations
  - unsuitable for optimization

Rotations in optimization - Euler angles

Euler angles $(\theta_x, \theta_y, \theta_z)$ use only three numbers

- Suffers from gimbal lock, i.e. that two parameters do the same in certain configurations
  - unsuitable for optimization
- If the rotation is close to identity, we are far from gimbal lock
  - no problems in this case
  - optimize over a rotation relative to the initial guess
  - $\theta_x, \theta_y, \theta_z$ will usually stay close to zero
    - only if initial guess is good
    - for finite difference start from $\theta_x = \theta_y = \theta_z = 2\pi$

Rotations in optimization - Axis-angle

Axis-angle represents a rotation as a rotation of θ around an axis $\boldsymbol{v}$, where $\|\boldsymbol{v}\|_2 = 1$.

Store as a vector $\boldsymbol{v}/\theta$ (only three elements).

How OpenCV returns rotations (`rvec`). Can be converted to a rotation matrix with `cv2.Rodrigues`.

Has singularity at $\mathbf{0}$, but works well in most cases.

Rotations in optimization - Quaternions

Quaternions are the gold standard for representing rotations.

A quaternion uses four numbers represent a rotation (q_1, q_2, q_3, q_4) subject to $q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$.

Smooth and suffers from no problems, however we have to normalize the quaternion each optimization step, to ensure it is still a valid quaternion.

Software packages

There are many implementations out there!

- Ceres (C++)
  - Dual numbers (no implementing derivatives, 😜)
  - Quaternion parameterization
- `scipy.optimize.least_squares`
  - Easy to use
  - Finite differences or analytical gradients

Camera calibration

Outline

More tips and tricks

- Checkerboard alternatives
- Subpixel estimation
- Overfitting
- Bundle adjustment

Checkerboard

OpenCV needs to see the all corners of the checkerboard in order to detect it.

Getting the entire image plane covered in detected points is hard, especially near the edges.

ChArUco

A ChArUco boards is a checkerboard with ArUco markers.

Possible to detect partial checkerboards!

Subpixel corner estimation

- Where is the corner?

Subpixel corner estimation

- Where is the corner?
- Look at a neighbourhood around a corner
- OpenCV has `cv2.cornerSubPix` but it is 💩
- What to do?

Good method for subpixel corner estimation

A good subpixel corner estimator is presented in Schops et al.[1]

Generate n random 2D vectors relative to the corner $\boldsymbol{s}_i$

$$C_{sym}(\boldsymbol{H}) = \sum_{i=1}^{n} \left(\left(I(\boldsymbol{H}(\boldsymbol{s}_i)) - I(\boldsymbol{H}(-\boldsymbol{s}_i)) \right)^2 \right)$$

$\boldsymbol{H}$ maps a point using the homography, and I interpolates the value of the image at the given point. Minimize C_{sym}.

Camera calibration - A cautionary tale

- Including more distortion parameters in your model will always give a lower re-projection error

Camera calibration - A cautionary tale

- Including more distortion parameters in your model will always give a lower re-projection error
- . . . for the images used to calibration!
  - Will not necessarily generalize.
- This can be overfitting
- Use cross-validation

Camera calibration - How to do cross validation

- For each checkerboard, split your detected corners into a validation and training set
- Do the calibration using all training corners
- Use the estimated checkerboard pose to reproject the validation corners

Bundle adjustment

- The checkerboard can have imperfections, such as not being flat
  - The checkerboards you get today are not very flat
- Optimize everything again, including the 3D positions of the corners

Learning objectives

After this lecture you should be able to:

- perform non-linear optimization within computer vision
- explain the principle behind Levenberg-Marquardt
- compute the Jacobian of a function
- reason about different parameterizations of rotations in optimization

Info about exercise and exercise time!

Simple features

Morten R. Hannemose, mohan@dtu.dk

March 10, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- explain the image correspondence problem
- explain the use of image features (key points, interest points)
- understand and implement image filtering
- implement Harris corner detection
- run Canny edge detection

Presentation topics

- Image correspondence problem
  - Features and feature descriptors
- Simple features
  - Filtering and convolution
  - Harris corners
  - Canny edges

Image correspondence problem

Image correspondence

The problem of matching two parts of different images:

Image correspondence

More images of the same scene

- Correspondence exists between parts that are visible in each image
- Not all corresponding points have a unique pattern
- Idea: Only choose points that can be identified uniquely

What are good points?

What are good points?

We are able to identify the same corners in multiple images

Problems

Movement of camera

- Scale – distance of camera to object
- Rotation – objects and camera is rotated between frames
- Translation – movement of camera from one place to another

Result: Appearance change dependent on distance to camera

Problems

Movement of camera

Problems

Other issues

- Occlusion – objects can be in front of other objects
- Lighting change – darker/lighter, color of light (change in spectrum), direction of light
- Clutter – many objects in a scene

Solution

Invariance to imaging problems

- Focus on points and a small area around each point
- Two aspects
  - identify key points (focus of today)
  - characterize pattern around point (week 8)

Features and feature descriptors

First some terminology:

Key points, interest points, and feature points are the same – namely points in an image with a coordinate position (also between pixels).

Descriptors, key point descriptors, interest point descriptors, and feature descriptors are the same and used for characterizing the pattern around a point. Usually a vector that can be matched between images.

The image patch feature descriptor

One easy way of describing a feature is to use the local pixel values around the feature.

The image patch matching

Stretch the image patch into a vector $\boldsymbol{f}$, and that is your descriptor.

A simple comparison operator $d(\boldsymbol{f}_1, \boldsymbol{f}_2)$ just uses the scalar product as the distance between features:

$$d(\boldsymbol{f}_1, \boldsymbol{f}_2) = \boldsymbol{f}_1^\top \boldsymbol{f}_2$$

Good with only small transformations, low noise, and unchanged environment.

Cross correlation

A better comparison $X(\boldsymbol{f}_1, \boldsymbol{f}_2)$ uses the cross correlation as the distance:

$$X(\boldsymbol{f}_1, \boldsymbol{f}_2) = \frac{\mathrm{cov}(\boldsymbol{f}_1, \boldsymbol{f}_2)}{\sqrt{\mathrm{var}(\boldsymbol{f}_1) + \mathrm{var}(\boldsymbol{f}_2)}}, \text{ where}$$

$$\mathrm{cov}(\boldsymbol{p}, \boldsymbol{q}) = \frac{1}{N-1} \sum_i^N (\boldsymbol{p}_i - \overline{\boldsymbol{p}})^\mathsf{T} (\boldsymbol{q}_i - \overline{\boldsymbol{q}}), \text{ and}$$

$$\mathrm{var}(\boldsymbol{p}) = \frac{1}{N-1} \sum_i^N ||\boldsymbol{p}_i - \overline{\boldsymbol{p}}||^2.$$

Cross correlation

- The cross correlation picks up local variation, and is therefore more robust against environment changes than squared distance
- Only useful when images are extremely similar.

More complicated feature descriptors will follow in two weeks.

Simple features

Filtering and convolution

$$I(x;t) = \int_{\xi \in \mathbb{R}^2} I(x-\xi)g(\xi;t)d\xi \approx \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x-i, y-j)g(i,j)$$

2D convolution

We use filtering and convolution for the same operation and use the symbol $*$ for convolution. Convolution is commutative

$$I_g = g * I = I * g$$

2D Gaussian $g : \mathbb{R}^2 \times \mathbb{R}_+ \to \mathbb{R}$

$$g(x, y; \sigma^2) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-(x^2 + y^2)}{2\sigma^2}\right). \tag{1}$$

Convolution – separability

Useful property: The (isotropic) Gaussian is separable

$$g(x, y; \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right) \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-y^2}{2\sigma^2}\right)$$

This separability means that

$$I_g = \left(\mathbf{g} * \mathbf{g}^\mathsf{T}\right) * I = \mathbf{g} * \left(\mathbf{g}^\mathsf{T} * I\right)$$

where $\mathbf{g}$ is a vector of the Gaussian.

Size of Gaussian filter

- Ideally: Infinitely big – but this is impractical
- Empiric rule: 3σ, 4σ or 5σ rule – example:
  - if $\sigma = 2$, filter size of Gaussian is $2 \cdot 5 \cdot 2 + 1 = 21$
  - if $\sigma = 20$, filter size of Gaussian is $2 \cdot 5 \cdot 20 + 1 = 201$

Demo – Gaussian filtering

`cv2.filter2D`

`cv2.sepFilter2D`

Derivatives of an image

How can we find the derivative of an image in e.g. the x-direction?

What does it mean to take a derivative?

Derivatives of an image

How can we find the derivative of an image in e.g. the x-direction?

What does it mean to take a derivative?

An image is only discrete values.

If we can make the image continuous all our math works again

Derivatives of an image using Gaussians

Let $\mathbf{g}$ be column vector of a Gaussian. Consider the blurred image

$$I_b = \mathbf{g} * \mathbf{g}^\top I.$$

Derivatives of an image using Gaussians

Let $\mathbf{g}$ be column vector of a Gaussian. Consider the blurred image

$$I_b = \mathbf{g} * \mathbf{g}^\mathsf{T} I.$$

Then the derivative of I_b in the x-direction is

$$\begin{aligned}\frac{\partial}{\partial x} I_b &= \frac{\partial}{\partial x}\left(\mathbf{g} * \mathbf{g}^\mathsf{T} * I\right) \\ &= \mathbf{g} * \left(\frac{\partial}{\partial x}\mathbf{g}^\mathsf{T}\right) * I \\ &= \mathbf{g} * \mathbf{g}_d^\mathsf{T} * I,\end{aligned}$$

Derivative of the Gaussian

Recall the one-dimensional Gaussian

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right).$$

We can then compute the derivative

$$g_d(x) = \frac{d}{dx} g(x) = \frac{-x}{\sigma^2} g(x).$$

This is straightforward to implement on a computer.

Short break

Harris corners

Corners and blobs are great features because they are easy to describe and detect.

Harris corners are detected through as points with locally maximum change from a small shift

$$\Delta I(x, y) = I(x, y) - I(x + \Delta x, y + \Delta y)$$

Harris corner measure

We define a corner as having a large $\Delta I(x, y)^2$, and we average the value over a small area to ensure robustness against noise.

The corner measure is

$$\begin{aligned} c(x, y) &= g * \Delta I(x, y)^2 \\ &= g * \big(I(x, y) - I(x + \Delta x, y + \Delta y)\big)^2, \end{aligned}$$

where $g * \ldots$ is the convolution with the Gaussian.

Replace with Taylor approximation

$$I(x + \Delta x, y + \Delta y) \approx I(x, y) - \frac{\partial I}{\partial x}(x, y)\,\Delta_x - \frac{\partial I}{\partial y}(x, y)\,\Delta_y$$

Replace with Taylor approximation

$$I(x+\Delta x, y+\Delta y) \approx I(x,y) - \underbrace{\frac{\partial I}{\partial x}(x,y)}_{I_x}\,\Delta_x - \underbrace{\frac{\partial I}{\partial y}(x,y)}_{I_y}\,\Delta_y$$

Replace with Taylor approximation

$$I(x+\Delta x, y+\Delta y) \approx I(x,y) - \underbrace{\frac{\partial I}{\partial x}(x,y)}_{I_x}\Delta_x - \underbrace{\frac{\partial I}{\partial y}(x,y)}_{I_y}\Delta_y$$

$$= I(x,y) - \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix}$$

I_x and I_y are also the values at (x, y), but we omit it for readability.

Harris corner measure derivation

The corner measure is then

$$
\begin{aligned}
c(x,y) &= g * \big(I(x,y) - I(x+\Delta x, y+\Delta y)\big)^2 \\
&\approx g * \left(\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix}\right)^2
\end{aligned}
$$

Harris corner measure derivation

The corner measure is then

$$
\begin{aligned}
c(x,y) &= g * \big(I(x,y) - I(x+\Delta x, y+\Delta y)\big)^2 \\
&\approx g * \left(\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix}\right)^2 \\
&= g * \left(\begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}\right) \\
&= g * \left(\begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}\right)
\end{aligned}
$$

Harris corner measure derivation

Finally, we take the convolution with the Gaussian inside

$$
\begin{aligned}
c(x,y) &\approx g * \left(\begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right) \\
&= \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \underbrace{\begin{bmatrix} g * (I_x^2) & g * (I_x I_y) \\ g * (I_x I_y) & g * (I_y^2) \end{bmatrix}}_{\boldsymbol{C}(x,y)} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}.
\end{aligned}
$$

The averaged Hessian

We end up with an averaged Hessian matrix $\boldsymbol{C}(x, y)$.

Corners have a large $c(x, y)$ regardless of the direction of $\begin{bmatrix}\Delta_x & \Delta_y\end{bmatrix}$.

$$c(x, y) = \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} \begin{bmatrix} a & c \\ c & b \end{bmatrix} \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}$$

When is this the case?

The averaged Hessian

We end up with an averaged Hessian matrix $\boldsymbol{C}(x, y)$.

Corners have a large $c(x, y)$ regardless of the direction of $\begin{bmatrix}\Delta_x & \Delta_y\end{bmatrix}$.

$$c(x, y) = \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} \begin{bmatrix} a & c \\ c & b \end{bmatrix} \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}$$

When is this the case?

When both eigenvalues of $\boldsymbol{C}(x, y)$ are large.

The Harris corner metric continued

Let λ_1 and λ_2 be the eigenvalues of $\boldsymbol{C}(x, y)$.
We then have the following

$$\det\big(\boldsymbol{C}(x,y)\big) = \lambda_1\lambda_2,$$
$$\operatorname{trace}\big(\boldsymbol{C}(x,y)\big) = \lambda_1 + \lambda_2, \text{ let us then define}$$
$$r(x,y) = \det\big(\boldsymbol{C}(x,y)\big) - k \ \operatorname{trace}\big(\boldsymbol{C}(x,y)\big)$$

The Harris corner metric

Now we have the Harris corner metric

$$r(x, y) = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2.$$

k is a free parameter, typically $k = 0.06$.

Notice that $r(x, y)$ is:

- negative for one eigenvalue much greater than the other
- large and positive for large eigenvalues
- small and positive for small eigenvalues

The Harris corner detector

We can now detect corners by finding points where $r(x, y)$ is greater than some threshold τ.

Typically, you can choose τ to be

$$0.1 \cdot \max\big(r(x, y)\big) < \tau < 0.8 \cdot \max\big(r(x, y)\big).$$

Non-maximum suppression

Find the local maximum of one pixel compared to neighbours

$$\big(I(x,y) - I(x',y')\big) > 0 \;\; \forall \;\; x' \in \mathcal{N}(x,y)$$

where $\mathcal{N}(x,y)$ is a neighbourhood around the point (x,y)

Suggested procedure

- Initialize an array of size of image with $I(x,y) > \tau$
- Compare entire image with one neighbour (e.g to the right)
- Set pixels to 0 where it is not larger than neighbour
  - Repeat for all neighbours
- Find coordinates of all pixels that are 1

Canny edges

Often, we might also consider detecting lines.

To detect (non-straight) lines, we can use the Canny edge detector.

Canny edges

The metric in the canny edge detector is simply the gradient magnitude

$$m(x, y) = \sqrt{I_x^2(x, y) + I_y^2(x, y)}.$$

And the edges are detected by thresholding the magnitude $m(x, y)$, however, in two stages: seed and grow.

Canny edges

- Seed: label edges by a threshold where $m(x, y) > \tau_1$
- Grow: with a second threshold where $m(x, y) > \tau_2$, label iif the new points are next to previously labelled edges.
- The threshold values are chosen such that $\tau_1 > \tau_2$

Exercise

During the exercise, you will

- implement the Harris corner detector
- try the Canny edge detector

Suggestion: After finishing your implementation – try on your own images and perhaps draw some features and see how it works

Learning objectives

After this lecture you should be able to:

- explain the image correspondence problem
- explain the use of image features (key points, interest points)
- understand and implement image filtering
- implement Harris corner detection
- run Canny edge detection

Exercise time!

Robust Model Fitting

Morten R. Hannemose, mohan@dtu.dk

March 16, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- explain how the Hough transform works
- understand and implement RANSAC

Presentation topics

Hough Transform

RANSAC

- Recap of lines in homogeneous coordinates

Fitting models

Can we fit a straight line?

Fitting models

Can we fit a straight line?

Yes we can!

Fitting models

Can we fit a straight line when there are a few outliers?

Fitting models

Can we fit a straight line when there are a few outliers?

Not really...
We get a very bad fit.
We need robust ways to fit models!

About lines

- This presentation will uses fitting straight lines to data for all examples.
- The same principles generalize to other models!

Hough Transform

Hough Transform

Is a transformation of an edge image where lines can be extracted.

Example image

Hough transform of image

Hough Transform

How to represent a line?

- $y = ax + b$?
  - Has singularities for vertical lines
- Homogeneous coordinates?
  - Is over-parametrized

Hough Transform - r, θ representation

Represent using:

- θ angle of the line
- r closest distance from origin to line

Closely related to the homogeneous line representation

$$\boldsymbol{l} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & -r \end{bmatrix}$$

Hough Transform

We can represent any line with these two parameters.

- $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}[$
- $r \in [-d, d]$, where d is the diagonal length of the image.

We discretize these values and represent them on a 2D grid

Hough Transform

Idea: Let points vote on which line is the best!

- Each point can be part of (infinitely) many lines.
- All potential lines going through this point are of interest
- Each point votes on all lines that go through its
  - This corresponds to a line in Hough space
- Repeat for all points

Hough Transform – Example

Image

Hough Space

Hough Transform – Example

Image

Hough Space

Hough Transform – Example

Image

Hough Space

Hough Transform – Example

Image

Hough Space

Hough Transform – Example

Image

Hough Space

Hough Transform

- A peak in Hough space corresponds to a line in the image.
- Can be found using non-maximum suppression.

Generalized Hough Transform

- We can generalize the Hough transform for more complex models.
  - e.g. for circles, the Hough space is now in 3D and each point becomes a conic.
- Hough space has same number of dimensions as the model we fit has degrees of freedom.
- Impractical for more than three degrees of freedom

RANSAC

Random sample consensus (RANSAC)

Idea!

Instead of computing the Hough space, what if we could sample points directly in Hough space?

Random sample consensus (RANSAC)

Idea!

Instead of computing the Hough space, what if we could sample points directly in Hough space?

What if we could sample with the value in hough space being proportional to the probability of sampling the point?

RANSAC

- Randomly sample the minimum number of points we need to fit our model
- Fit the model to these samples

Does this line fit the data well?

Measure inliers

Points closer than a certain threshold to the line are inliers!

Number of inliers is indication of how well the line fits.

Example line

Example line

Example line

Example line

The RANSAC algorithm

Keep track of which fit had the most inliers so far, and update this if a fit with more inliers is found

The RANSAC algorithm

- Sample minimum number of points required to fit model
  - Fit model to these
- Data points with an error less than τ are inliers with respect to fitted model
  - If number of inliers is higher than the highest number of inliers seen so far, update best model.

The RANSAC algorithm

- Sample minimum number of points required to fit model
  - Fit model to these
- Data points with an error less than τ are inliers with respect to fitted model
  - If number of inliers is higher than the highest number of inliers seen so far, update best model.
- Repeat for N iterations.
- Final step:
  - Re-fit model to all inliers of the best model

Implementation details

- Represent the lines using homogeneous coordinates $\begin{bmatrix} a & b & c \end{bmatrix}$
  - Makes it easy to compute distance to line
  - Scale the line such that $a^2 + b^2 = 1$
- Recall that distance to line is given by: $|\boldsymbol{l} \cdot \Pi^{-1}(\boldsymbol{p})|$
  - When $a^2 + b^2 = 1$

RANSAC

- Sample in Hough space without computing it.
- Useful for fitting models when outliers are present.
- We must select the threshold for inliers and the number of iterations carefully.
- Being able to fit a model to the least amount of data points is of interest

How many iterations?

- We can come with some idea of how many iterations we need
- Assume fraction of outliers is ϵ
- i.e. $\epsilon = 0.1$ means 10% of data are outliers.

How many iterations?

- We need n data points to fit a single model

$$P(\text{one sample has only inliers}) = (1-\epsilon)^n$$

- Set p to a high value such as $p = 0.99$. Useful if we know ϵ.

How many iterations?

- We need n data points to fit a single model

$$P(\text{one sample has only inliers}) = (1-\epsilon)^n$$
$$P(\text{one sample has one or more outliers}) = 1-(1-\epsilon)^n$$

- Set p to a high value such as $p = 0.99$. Useful if we know ϵ.

How many iterations?

- We need n data points to fit a single model

$$P(\text{one sample has only inliers}) = (1-\epsilon)^n$$
$$P(\text{one sample has one or more outliers}) = 1-(1-\epsilon)^n$$
$$P(N \text{ samples all have one or more outliers}) = (1-(1-\epsilon)^n)^N$$

- Set p to a high value such as $p = 0.99$. Useful if we know ϵ.

How many iterations?

- We need n data points to fit a single model

$$P(\text{one sample has only inliers}) = (1-\epsilon)^n$$
$$P(\text{one sample has one or more outliers}) = 1-(1-\epsilon)^n$$
$$P(N \text{ samples all have one or more outliers}) = (1-(1-\epsilon)^n)^N$$
$$P(\text{at least one of } N \text{ samples has only inliers}) = 1-(1-(1-\epsilon)^n)^N$$

- Set p to a high value such as $p = 0.99$. Useful if we know ϵ.

How many iterations?

- We need n data points to fit a single model

$$
\begin{aligned}
P(\text{one sample has only inliers}) &= (1-\epsilon)^n \\
P(\text{one sample has one or more outliers}) &= 1-(1-\epsilon)^n \\
P(N \text{ samples all have one or more outliers}) &= (1-(1-\epsilon)^n)^N \\
P(\text{at least one of } N \text{ samples has only inliers}) &= 1-(1-(1-\epsilon)^n)^N = p \\
\Leftrightarrow N &= \frac{\log(1-p)}{\log\left((1-(1-\epsilon)^n)\right)}
\end{aligned}
$$

- Set p to a high value such as $p = 0.99$. Useful if we know ϵ.

Determining number of iterations adaptively

- We can estimate an upper bound of ϵ while running RANSAC.
- Let be s the number of inliers in the best model found so far
- Let be m the total number of datapoints
- $\hat{\epsilon} = 1 - \frac{s}{m}$

Determining number of iterations adaptively

- We can estimate an upper bound of ϵ while running RANSAC.
- Let be s the number of inliers in the best model found so far
- Let be m the total number of datapoints
- $\hat{\epsilon} = 1 - \frac{s}{m}$
- We can now estimate an upper bound on the number of iterations required
- $\hat{N} = \frac{\log(1-p)}{\log\left((1-(1-\hat{\epsilon})^n)\right)} > N$
- Terminate once we have done more than $\hat{N}$ iterations.

Recap of lines in homogeneous coordinates

- How do we fit a line to two points ($\mathbf{p}_1$ and $\mathbf{p}_2$) in homogeneous coordinates?
- What do we know about the line $\mathbf{l}$?
- $\mathbf{l} \cdot \mathbf{p}_1 = 0$ and $\mathbf{l} \cdot \mathbf{p}_2 = 0$.

Recap of lines in homogeneous coordinates

- How do we fit a line to two points ($\mathbf{p}_1$ and $\mathbf{p}_2$) in homogeneous coordinates?
- What do we know about the line $\mathbf{l}$?
- $\mathbf{l} \cdot \mathbf{p}_1 = 0$ and $\mathbf{l} \cdot \mathbf{p}_2 = 0$.
- The dot product is zero only when two vectors are perpendicular.
- How can we find $\mathbf{l}$ such that it is perpendicular to $\mathbf{p}_1$ and $\mathbf{p}_2$?

Recap of lines in homogeneous coordinates

- How do we fit a line to two points ($\mathbf{p}_1$ and $\mathbf{p}_2$) in homogeneous coordinates?
- What do we know about the line $\mathbf{l}$?
- $\mathbf{l} \cdot \mathbf{p}_1 = 0$ and $\mathbf{l} \cdot \mathbf{p}_2 = 0$.
- The dot product is zero only when two vectors are perpendicular.
- How can we find $\mathbf{l}$ such that it is perpendicular to $\mathbf{p}_1$ and $\mathbf{p}_2$?
- Use the cross product! $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$.

Learning objectives

After this lecture you should be able to:

- explain how the Hough transform works
- understand and implement RANSAC

Midterm Evaluation

Exercise time!

Blobs and SIFT features

Morten R. Hannemose, mohan@dtu.dk

March 24, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- implement and use blob detection using Difference-of-Gaussians
- analyse and use SIFT features and feature matching

Similarity

Pixel-wise comparison

- Shift of single pixel vs. two views

Similarity

Basic idea

- Locally appearance between views is the same
- Variation can be handled via invariances

Local image features

SIFT – key elements

- Features localized at interest points
- Adapted to scale and invariant to appearance changes

SIFT – scale invariant feature transform (Lowe, 2004)

- Scale-space blob detection – difference of Gaussians
- Interest point localization
- Orientation assignment
- Interest point descriptor
- Note – SIFT is one example of interest point feature

Harris corners and BLOBs

Harris corners are features that have a large change of intensity in two orthogonal directions. They are:

- local,
- can be found at different scales by changing the Gaussian filters, and
- invariant to rotation.

Harris corners are found by first order derivatives whereas blobs are response to second order image derivatives.

BLOBs – Binary Large OBjects

Correspond to:

- a dark area surrounded by brighter intensities or,
- a bright area surrounded by darker intensities

Hessian

The Hessian matrix contains the second order derivatives

$$\mathbf{H}(x,y) = \begin{bmatrix} I_{xx}(x,y) & I_{xy}(x,y) \\ I_{xy}(x,y) & I_{yy}(x,y) \end{bmatrix},$$

where

$I_{xx}(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2}$, $I_{yy}(x,y) = \frac{\partial^2 I(x,y)}{\partial y^2}$, and $I_{xy}(x,y) = \frac{\partial^2 I(x,y)}{\partial x \partial y}$.

Curvature

Second order derivatives measure curvature.

Eigenvalues of the Hessian (λ_1, λ_2) measure the principal curvature, i.e. degree of change in intensity

Eigenvectors measure direction of change where the eigenvector corresponding to the largest eigenvalue (λ_1) is the direction of most change and the second is orthogonal to that.

Blob detection with Hessian

Similar to the Harris corner detector, we can use either of the measures

$$\det(\mathbf{H}) = \lambda_1 \lambda_2,$$
$$\mathrm{trace}(\mathbf{H}) = \lambda_1 + \lambda_2,$$

where λ_i are the eigenvalues of the Hessian.

$\det(\mathbf{H})$ is the Gaussian curvature.

$\mathrm{trace}(\mathbf{H}) = \nabla^2 I$ is the Laplacian, which we use for blob detection.

The Laplacian

Two Gaussians with different variance

Difference of Gaussians vs Laplacian

Blob detection with DoG

The Laplacian $\nabla^2 I$ can be approximated with the Difference-of-Gaussians (DoG).

Blurring the image with two different Gaussian kernels:

$$\nabla^2 I \approx D_\sigma = (G_{k\sigma} - G_\sigma) * I = G_{k\sigma} * I - G_\sigma * I,$$

where G_σ is a Gaussian with standard deviation σ and $k > 1$ is a scale factor.

Blob detection with DoG

The Laplacian $\nabla^2 I$ can be approximated with the Difference-of-Gaussians (DoG).

Blurring the image with two different Gaussian kernels:

$$\nabla^2 I \approx D_\sigma = (G_{k\sigma} - G_\sigma) * I = G_{k\sigma} * I - G_\sigma * I,$$

where G_σ is a Gaussian with standard deviation σ and $k > 1$ is a scale factor.

Why are we interested in this approximation?

Blob detection with DoG

Figure 1: A dog detecting blobs

SIFT – Scale invariance

SIFT – Scale invariance

- Using difference of Gaussians for blob detection

$$
\begin{aligned}
D(x, y, \sigma) &= \big((G_{k\sigma} - G_\sigma) * I\big)(x, y) \\
&= (G_{k\sigma} * I)(x, y) - (G_\sigma * I)(x, y) \\
&= L(x, y, k\sigma) - L(x, y, \sigma)
\end{aligned}
$$

The DoG is computed by subtracting more and and more blurred images from each other.

SIFT - Difference of Gaussians

Gaussian scale space – Efficient

- Convolution of two Gaussians yield a new Gaussian
- Generate scale space by iteratively blurring already blurred images again
  - Otherwise we would need very large Gaussian kernels
- However the size of the kernel still grows
  - $(k\sigma)^2 = \sigma^2 + new^2$

Gaussian scale space – Efficient

- Convolution of two Gaussians yield a new Gaussian
- Generate scale space by iteratively blurring already blurred images again
  - Otherwise we would need very large Gaussian kernels
- However the size of the kernel still grows
  - $(k\sigma)^2 = \sigma^2 + new^2$
- We choose $k = 2^{\frac{1}{3}}$
- σ doubles after three images, the image is downsampled.
  - This is an octave.
  - We only need three Gaussians of constant size (precomputed)

SIFT – Estimation of DoG

Difference of Gaussians

Extrema localization

SIFT - Magnitude of the DoG response

Do we get smaller values in the difference of Gaussians for high values of σ?

Using the heat equation it can be shown that the response does not change as a function of sigma.

We can use the same threshold for the entire scale space.

SIFT – Subpixel localization

- Why is this relevant?

SIFT – Subpixel localization

- Why is this relevant?
- Second order Taylor approximation of DoG around local maximum:

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

- Setting the derivative of $D(\mathbf{x})$ to zero

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1}\frac{\partial D}{\partial \mathbf{x}}$$

SIFT – Subpixel localization

- We get

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}}$$

- If $|D(\hat{\mathbf{x}})| < 0.03$ the point is discarded
  - Removes points with low contrast.

SIFT – Interest point along edges discarded

- The eigenvalues of the Hessian are proportional to the principal curvatures

$$\mathbf{H} = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix}$$

Interest point along edges discarded

- λ_1 and λ_2 are the eigenvalues of the Hessian

$$\begin{aligned} \mathrm{trace}(\mathbf{H}) &= I_{xx} + I_{yy} = \alpha + \beta \\ \det(\mathbf{H}) &= I_{xx} I_{yy} - I_{xy}^2 = \alpha\beta \end{aligned}$$

Points are kept if

$$\frac{\mathrm{trace}(\mathbf{H})^2}{\det(\mathbf{H})} < \frac{(r+1)^2}{r},$$

where $r = 10$ (found to be a good heuristic)

Short break

DoG measure

Features have $|D(\boldsymbol{x})| > \tau$, where τ is a threshold.

Dark BLOBs: $D(\boldsymbol{x}) > 0$, bright BLOBs: $D(\boldsymbol{x}) < 0$

DoG in scale pyramids

Scale pyramids are increasingly blurred of the same image.
Scale space DoG is subtraction between all layers adjacent scales.

Scale space blobs and DoG

DoG at different scales makes for a scale invariant feature detector.

Small and large details are recoverable in different DoGs.

SIFT – Orientation assignment

Compute the orientation of gradients in a small region around the BLOB.

$$m(x, y) = \sqrt{L_x^2 + L_y^2}$$
$$\theta(x, y) = \arctan 2(L_y, L_x)$$

Where

$$L_x = L(x + 1, y) - L(x - 1, y)$$
$$L_y = L(x, y + 1) - L(x, y - 1)$$

SIFT – Orientation assignment

- Compute circular histogram of gradient orientations
  - Weighted by magnitude, smoothed, and has 36 bins

SIFT – Orientation assignment

- Compute circular histogram of gradient orientations
  - Weighted by magnitude, smoothed, and has 36 bins
- Use peak in histogram to assign orientation of point
- This introduces rotation invariance.
- Can we have multiple peaks in histogram?

SIFT – Orientation assignment

- Compute circular histogram of gradient orientations
  - Weighted by magnitude, smoothed, and has 36 bins
- Use peak in histogram to assign orientation of point
- This introduces rotation invariance.
- Can we have multiple peaks in histogram?
  - Yes, this can happen at e.g. corners.
  - Create a new point at the same location if peak is over 80% of max.

SIFT – Descriptor

- Create local patch at scale and orientation of point
- Build a histogram of local gradient orientations

- Normalized using L_2 norm: $\mathbf{d}_n = \frac{1}{\sqrt{\sum_{i=1}^{128} \mathbf{d}(i)^2}} \mathbf{d}$

Taylor SIFT

Taylor SIFT

SIFT – Invariances

- Position
- Scale
- Rotation
- Linear intensity change
- Perspective changes?

SIFT – Matching of descriptors

- Use Euclidean distance between normalized vectors

$$\delta(\mathbf{d}_i, \mathbf{d}_j) = \sqrt{\sum_{n=1}^{128} (d_{i,n} - d_{j,n})^2}$$

- Note – for comparison the square root is not needed

RootSIFT

Simple trick to improve SIFT matching

- SIFT is a histogram
- Euclidean distance is dominated by large values
- RootSIFT is a transformation that measures distance using the Hellinger kernel.
  - L1 normalize
  - Take the square root of each element
  - L2 normalize the resulting vector
- Compare using Euclidean distance

SIFT – Matching of descriptors

- For each feature in image 1 ($\boldsymbol{d}_{1,i}$) find the closest feature in image 2 ($\boldsymbol{d}_{2,j}$)
  - This will give a lot of incorrect matches
- Cross checking
  - Only keep matches where $\boldsymbol{d}_{2,j}$ is also the closest to $\boldsymbol{d}_{1,i}$ of all features in image 1

SIFT – Matching of descriptors

- For each feature in image 1 ($\boldsymbol{d}_{1,i}$) find the closest feature in image 2 ($\boldsymbol{d}_{2,j}$)
  - This will give a lot of incorrect matches
- Cross checking
  - Only keep matches where $\boldsymbol{d}_{2,j}$ is also the closest to $\boldsymbol{d}_{1,i}$ of all features in image 1
- Ratio test
  - Compute the ratio between the closest and second closest match, and keep where this is below a threshold, e.g. 0.7.

SIFT – Summary

- SIFT is both a feature detector and descriptor
- Find local extrema of DoGs in scale space
- Place patch oriented along local gradients
- Compute histograms of gradients.
- Allows matching of images invariant to: scale, rotation, illumination and viewpoint
- Partly visible objects can be matched

Other descriptors

- SIFT is widely used.
  - Was patented until 2020.
- Meanwhile other similar methods were created
  - ORB
  - SURF
  - BRISK
  - FAST

Learned descriptors

- Deep Learning has created improved feature descriptors.
- Mostly in improvement in invariance to changing lighting.
- Some examples:
  - R2D2: Repeatable and Reliable Detector and Descriptor
    https://github.com/naver/r2d2
  - Superpoint
    https://github.com/magicleap/SuperPointPretrainedNetwork

Learning objectives

After this lecture you should be able to:

- implement and use blob detection using Difference-of-Gaussians
- analyse and use SIFT features and feature matching

Exercise

Build a blob detector and match points with SIFT detector.

Python: Use OpenCV (4.2.0 or newer)

Matlab: Use VLFeat
https://www.vlfeat.org/overview/sift.html

Exercise time!

Geometry Constrained Feature Matching

Morten R. Hannemose, mohan@dtu.dk

March 30, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- explain and implement the eight point algorithm for estimating the fundamental matrix
- explain and implement estimation of the fundamental matrix with RANSAC
- choose the threshold for RANSAC using χ^2

Presentation topics

- Estimating the fundamental matrix
  - Linear algorithm
  - Incorporating RANSAC
- Thresholding for RANSAC

Setting the scene

- Stereo geometry
- (SIFT) Features
- RANSAC

Geometry constrained feature matching

Use multi view geometry to filter matches

Fundamental matrix – Recap

$\boldsymbol{R}$ and $\boldsymbol{t}$ describe the relative pose between the cameras.

$$\boldsymbol{E} = [\boldsymbol{t}]_\times \boldsymbol{R}$$
$$\boldsymbol{F} = \boldsymbol{K}_2^{-\top} \boldsymbol{E} \boldsymbol{K}_1^{-1}$$
$$0 = \boldsymbol{q}_2^\top \boldsymbol{F} \boldsymbol{q}_1$$

The fundamental matrix expresses that corresponding points lie on their epipolar lines.

Estimating the fundamental matrix

Fundamental matrix problem

The fundamental matrix is defined with the relation $\boldsymbol{q}_2^\mathsf{T}\boldsymbol{F}\boldsymbol{q}_1 = 0$.

Consider the points $\boldsymbol{q}_{1i}$ and $\boldsymbol{q}_{2i}$ projected into cameras one and two, respectively. The relation is then

$$
\begin{aligned}
0 &= \boldsymbol{q}_{2i}^\mathsf{T}\boldsymbol{F}\boldsymbol{q}_{1i} \\
&= \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix}^\mathsf{T} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} x_{1i} \\ y_{1i} \\ 1 \end{bmatrix}
\end{aligned}
$$

Fundamental matrix problem

Rearrange the terms:

$$
\begin{aligned}
0 &= \boldsymbol{q}_{2i}^{\mathsf{T}} \boldsymbol{F} \boldsymbol{q}_{1i} \,, \\
0 &= \boldsymbol{B}^{(i)} \mathsf{flatten}(\boldsymbol{F}) \,,
\end{aligned}
$$

where

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

Fundamental matrix solution

Define $\boldsymbol{B}$

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

Subject to $||\text{flatten}(\boldsymbol{F})||_2 = 1$ the solution is the singular vector with the smallest singular value.

Degrees of freedom – Eight point algorithm

$\boldsymbol{F}$ has 9 numbers, and is scale invariant.

Each pair of corresponding point fixes a degree of freedom. Eight points is enough to estimate the fundamental matrix.

This is the eight point algorithm.

Degrees of freedom – Seven point algorithm

$[\boldsymbol{t}]_\times$ has rank 2, and thus $\boldsymbol{F}$ is also rank deficient, i.e. $\det(\boldsymbol{F}) = 0$.

Thus $\boldsymbol{F}$ has 7 degrees of freedom, and can be found from 7 matches.

$\boldsymbol{B}$ will have two singular vectors with singular value 0.
Denote these $\boldsymbol{F}'$ and $\boldsymbol{F}^\dagger$.

$\boldsymbol{F} = \alpha \boldsymbol{F}' + (1-\alpha)\boldsymbol{F}^\dagger$, where α is chosen such that $\det(\boldsymbol{F}) = 0$.

This is the seven point algorithm.

F can be estimated from eight point correspondences easily.

Possible to estimate from just seven.

Can be followed by non-linear optimization

Estimating the fundamental matrix with RANSAC

To use RANSAC, we need a way to measure distance from our model

What does $\boldsymbol{q}_{2i}^{\top}\boldsymbol{F}\boldsymbol{q}_{1i}$ equal for non corresponding points?

Estimating the fundamental matrix with RANSAC

To use RANSAC, we need a way to measure distance from our model

What does $\boldsymbol{q}_{2i}^{\mathsf{T}}\boldsymbol{F}\boldsymbol{q}_{1i}$ equal for non corresponding points?

$\boldsymbol{q}_{2i}^{\mathsf{T}}\boldsymbol{F}$ and $\boldsymbol{F}\boldsymbol{q}_{1i}$ are epipolar lines.

Distance from point to line

$$d = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x & y & 1 \end{bmatrix}^{\mathsf{T}}$$

should have $a^2 + b^2 = 1$, this is not guaranteed for the epipolar lines.

Symmetric epipolar distance

We want to measure the distance

- from $\boldsymbol{q}_{2i}$ to $\boldsymbol{F}\boldsymbol{q}_{1i}$ and
- from $\boldsymbol{q}_{1i}$ to $\boldsymbol{F}^\top\boldsymbol{q}_{2i}$

We can compute these as

$$\frac{\boldsymbol{q}_{2i}^\top\boldsymbol{F}\boldsymbol{q}_{1i}}{\sqrt{(\boldsymbol{F}\boldsymbol{q}_{1i})_1^2 + (\boldsymbol{F}\boldsymbol{q}_{1i})_2^2}} \quad \text{and} \quad \frac{\boldsymbol{q}_{2i}^\top\boldsymbol{F}\boldsymbol{q}_{1i}}{\sqrt{(\boldsymbol{q}_{2i}^\top\boldsymbol{F})_1^2 + (\boldsymbol{q}_{2i}^\top\boldsymbol{F})_2^2}}$$

where $\boldsymbol{x}_i^2$ refers to the square of th i^{th} element of $\boldsymbol{x}$.

Symmetric epipolar distance

We can normalize the distance to both epipolar lines, using their first two coordinates.

The squared symmetric epipolar distance is then given by

$$= \left(\boldsymbol{q}_{2i}^{\mathsf{T}} \boldsymbol{F} \boldsymbol{q}_{1i}\right)^2 \left(\frac{1}{(\boldsymbol{q}_{2i}^{\mathsf{T}} \boldsymbol{F})_1^2 + (\boldsymbol{q}_{2i}^{\mathsf{T}} \boldsymbol{F})_2^2} + \frac{1}{(\boldsymbol{F} \boldsymbol{q}_{1i})_1^2 + (\boldsymbol{F} \boldsymbol{q}_{1i})_2^2} \right),$$

Sampson's Distance

A similar distance is Sampson's distance.

$$d_{\mathsf{Samp}}(\boldsymbol{F}, \boldsymbol{q}_{1i}, \boldsymbol{q}_{2i}) = \frac{\left(\boldsymbol{q}_{2i}^{\top}\boldsymbol{F}\boldsymbol{q}_{1i}\right)^2}{(\boldsymbol{q}_{2i}^{\top}\boldsymbol{F})_1^2 + (\boldsymbol{q}_{2i}^{\top}\boldsymbol{F})_2^2 + (\boldsymbol{F}\boldsymbol{q}_{1i})_1^2 + (\boldsymbol{F}\boldsymbol{q}_{1i})_2^2}$$

Performs slightly better than the geometric distance in practice.

Is a squared distance.

Thresholding for RANSAC

Thresholding distances

- How to choose the threshold for RANSAC?

Thresholding distances

- How to choose the threshold for RANSAC?
- Introduce assumptions! 🙂
- Assume that the errors of inliers follow a normal distribution.

Dimensionality of the error

- Fundamental/essential matrix
  - The error is the distance to the epipolar line
  - This is an error in one dimension
- Homography
  - The error is the distance from mapped point to true point
  - This is an error in two dimensions
- Pose estimation/camera calibration
  - Again it is a distance between points
  - This is an error in two dimensions

This is called the codimension of the problem.

We denote this m.

Choosing the threshold

- Assume that the error of each sample follows an m-dimensional normal distribution with standard deviation σ
- The squared error is χ^2_m distributed (by definition)
- Always work with squared distances
  - Not necessary to take square root when comparing distances

Choosing the threshold

- Assume that the error of each sample follows an m-dimensional normal distribution with standard deviation σ
- The squared error is χ^2_m distributed (by definition)
- Always work with squared distances
  - Not necessary to take square root when comparing distances
- Choose a confidence interval e.g. 95%
  - i.e. we want our threshold to correctly identify 95% of all inliers.
- Look up the CDF for our χ^2_m distribution.
- E.g. for a fundamental matrix and 95%, $\tau^2 = 3.84 \cdot \sigma^2$

Some values from the CDF of χ^2_m

$m \backslash 1-p$	90%	95%	99%
1	2.71	3.84	6.63
2	4.61	5.99	9.21

Learning objectives

After this lecture you should be able to:

- explain and implement the eight point algorithm for estimating the fundamental matrix
- explain and implement estimation of the fundamental matrix with RANSAC
- choose the threshold for RANSAC using χ^2

Exercise time!

Image Stitching

Morten R. Hannemose, mohan@dtu.dk

April 13, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- explain and implement RANSAC to fit homographies
- understand explain the challenges involved in stitching panoramas

Presentation topics

Image stitching

Finding inliers

Transforming images

Multiple images

Nodal point

Setting the scene

Today you will be using:

- Homographies (hest)
- (SIFT) Features
- RANSAC

Image stitching

Image stitching – Panorama

About homographies

- What does a homography describe?

About homographies

- What does a homography describe?
  - The mapping of points between to images viewing the same plane.
- BUT! the mountains on the previous slide are not a plane!?
  - Why can we still use a homography?

Homographies for panoramas

- When the camera doesn't move but only rotates there are no perspective deformations
- It is equivalent to looking at a painting of the world
- Therefore we can assume the world is flat and use a homography!

Homographies for panoramas

- When the camera doesn't move but only rotates there are no perspective deformations
- It is equivalent to looking at a painting of the world
- Therefore we can assume the world is flat and use a homography!
- What happens if the camera moves?

Finding inliers

Measuring the error of a match

Let $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ be matching points without errors, such that

$$\begin{aligned}
\boldsymbol{p}_1 &= \boldsymbol{H}\boldsymbol{p}_2 \\
\boldsymbol{p}_1 &= \Pi^{-1}(\boldsymbol{q}_1), \quad \boldsymbol{p}_2 = \Pi^{-1}(\boldsymbol{q}_2) \\
\boldsymbol{q}_1 &= \Pi\left(\boldsymbol{H}\Pi^{-1}(\boldsymbol{q}_2)\right)
\end{aligned}$$

Measuring the error of a match

We have some 2D error in our detection of the points

$$\tilde{\boldsymbol{q}}_1 = \boldsymbol{q}_1 + \boldsymbol{\epsilon}_1, \quad \tilde{\boldsymbol{q}}_2 = \boldsymbol{q}_2 + \boldsymbol{\epsilon}_2$$
$$\tilde{\boldsymbol{p}}_1 = \Pi^{-1}(\tilde{\boldsymbol{q}}_1), \quad \tilde{\boldsymbol{p}}_2 = \Pi^{-1}(\tilde{\boldsymbol{q}}_2)$$

Measuring the error of a match

We have some 2D error in our detection of the points

$$\tilde{\boldsymbol{q}}_1 = \boldsymbol{q}_1 + \boldsymbol{\epsilon}_1, \quad \tilde{\boldsymbol{q}}_2 = \boldsymbol{q}_2 + \boldsymbol{\epsilon}_2$$
$$\tilde{\boldsymbol{p}}_1 = \Pi^{-1}(\tilde{\boldsymbol{q}}_1), \quad \tilde{\boldsymbol{p}}_2 = \Pi^{-1}(\tilde{\boldsymbol{q}}_2)$$

The error is the distance to the observed point in both images.

$$\left\|\tilde{\boldsymbol{q}}_1 - \Pi\left(\boldsymbol{H}\Pi^{-1}(\boldsymbol{q}_2)\right)\right\|_2^2 + \|\tilde{\boldsymbol{q}}_2 - \boldsymbol{q}_2\|_2^2$$

Do we know $\boldsymbol{q}_2$?

Measuring the error of a match

We have some 2D error in our detection of the points

$$\tilde{\boldsymbol{q}}_1 = \boldsymbol{q}_1 + \boldsymbol{\epsilon}_1, \quad \tilde{\boldsymbol{q}}_2 = \boldsymbol{q}_2 + \boldsymbol{\epsilon}_2$$
$$\tilde{\boldsymbol{p}}_1 = \Pi^{-1}(\tilde{\boldsymbol{q}}_1), \quad \tilde{\boldsymbol{p}}_2 = \Pi^{-1}(\tilde{\boldsymbol{q}}_2)$$

The error is the distance to the observed point in both images.

$$\left\|\tilde{\boldsymbol{q}}_1 - \Pi\left(\boldsymbol{H}\Pi^{-1}(\boldsymbol{q}_2)\right)\right\|_2^2 + \|\tilde{\boldsymbol{q}}_2 - \boldsymbol{q}_2\|_2^2$$

Do we know $\boldsymbol{q}_2$?

No!

Measuring the error of a match

We can use optimization to estimate the error free point

$$\mathrm{dist}^2_{\mathsf{true}} = \min_{\boldsymbol{q}_2} \left\| \tilde{\boldsymbol{q}}_1 - \Pi\left(\boldsymbol{H}\Pi^{-1}(\boldsymbol{q}_2)\right) \right\|_2^2 + \left\| \tilde{\boldsymbol{q}}_2 - \boldsymbol{q}_2 \right\|_2^2$$

Requires solving a least squares problem for each distance computation.

Impractical!

Practical approximation

Map the observed points back and forth, and compute the distance

$$\text{dist}^2_{\text{approx}} = \|\tilde{\boldsymbol{q}}_1 - \Pi\left(\boldsymbol{H}\tilde{\boldsymbol{p}}_2\right)\|_2^2 + \left\|\tilde{\boldsymbol{q}}_2 - \Pi\left(\boldsymbol{H}^{-1}\tilde{\boldsymbol{p}}_1\right)\right\|_2^2$$

Use this distance to measure if a pair of points are an inliers with respect to a homography.

Transforming images

Example – Images

Example – Matches

How to use the homography?

- We have estimated the homography
- How do we use it?
- Same approach as image undistortion.

Transforming images

- We can use the homography to warp an image to our current field of view.
- Generate all x, y coordinates for all pixels in the reference image
- Map these to the other image using the homography
- Use bilinear interpolation to compute the value at the transformed pixel locations.
  - Code is provided in the exercise.

Warping

- You have to decide in which area to evaluate the homography
- Simple, evaluate all homographies in the same area
  - Wasteful computation
  - Simple to implement
- Warp only the valid part of each image and move the images around
  - Necessary for larger panoramas

Example – Warped to same coordinate system

Note that x is negative.

Compositing images

- Overlap
- Average
- Median
- Graph cut

Example – Composited

Compositing – Average

Compositing – Graph Cut

Multiple images

Multiple images

Multiple images – Final result

How do we handle more than two images?

Example with three images:

Find the homographies between each pair $\boldsymbol{H}_{1\to 2}$ and $\boldsymbol{H}_{2\to 3}$.

Products of homographies are new homographies!

$$\boldsymbol{H}_{1\to 3} = \boldsymbol{H}_{1\to 2}\boldsymbol{H}_{2\to 3}$$

This principle extends to as many images as desired.

Multiple images

- Choose composting surface
  - I.e. which image is our baseline
  - Easy option is to choose one image to be the center
- Joint optimization

Nodal point

Capturing your own images

Which point should you actually rotate around?

- Image sensor?
- Pinhole?

Capturing your own images

Which point should you actually rotate around?

- Image sensor?
- Pinhole?

Real world camera

Which point should you actually rotate around?

Real world camera

Which point should you actually rotate around?

It depends on the camera, usually the middle of the lens.

What happens if we rotate around the wrong point?

Real world demo[ish]

And more!

- Known focal length reduces degrees of freedom
- Other projection models
- Exposure/color correction

Learning objectives

After this lecture you should be able to:

- explain and implement RANSAC to fit homographies
- understand explain the challenges involved in stitching panoramas

Exercise time!

Visual Odometry

Morten R. Hannemose, mohan@dtu.dk

April 21, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- choose the correct decomposition of the essential matrix
- explain why the scale of the translation is unknown
- explain the Perspective-n-Point problem
- implement a simple visual odometry algorithm

Presentation topics

Decomposing the Essential Matrix

Perspective-n-Point

Putting it all together

Motivation

Figure out how you are moving through the world by using a camera has many applications

- Drones
- Robotic vacuum cleaners
- Virtual reality headsets
- Augmented reality apps
- Autonomous cars

Many similar and related concepts

- Visual Odometry
- SLAM (Simultaneous Localization and Mapping)
- SfM (Structure from motion)

They all deal with some form of estimating a 3D map of the world and camera poses, but have emphasis on different parts.

Multiple “cameras”

The unknown scale of t – Mathematical argument

$$\boldsymbol{E} = \boldsymbol{R}[\boldsymbol{t}]_\times$$
$$0 = \boldsymbol{E}(s\boldsymbol{t}) = \boldsymbol{R}[\boldsymbol{t}]_\times(s\boldsymbol{t})$$

We can see that $\boldsymbol{t}$ lies in the null space of $\boldsymbol{E}$ but also that it can be arbitrarily scaled

The unknown scale of t – Conceptual reason

Decomposing the Essential Matrix

Essential matrix

You have matched features between two cameras, and want to make it robust.

Estimate $\boldsymbol{F}$ or $\boldsymbol{E}$ with RANSAC.

Estimating E

- How many points are required?
- Ask yourself:
  - How many degrees of freedom does it have?
  - How many degrees of freedom does a single point fix?

Estimating E

- How many points are required?
- Ask yourself:
  - How many degrees of freedom does it have?
  - How many degrees of freedom does a single point fix?
- Five.
- Not possible with linear algorithm from five points.
- Typically estimated using Nister's five-point algorithm
  - Involves solving tenth degree polynomial
  - Is implemented in OpenCV.

Decomposing the Essential Matrix

- The essential matrix can be computed from $\boldsymbol{R}$ and $\boldsymbol{t}$.
  - Can we recover them from $\boldsymbol{E}$?

Decomposing the Essential Matrix

- The essential matrix can be computed from $\boldsymbol{R}$ and $\boldsymbol{t}$.
  - Can we recover them from $\boldsymbol{E}$?
- Decomposing the essential matrix is ill posed
  - Two possible rotations
  - The sign of the translation is unknown.
- A total of four possible poses for the second camera
  - Check all four combinations
  - Choose the one with the most points in front of both cameras.

Decomposing the Essential Matrix

Back to visual odometry

We can find the pose of two cameras relative to each other

- How can we estimate the pose of a third camera?
- Using the essential matrix again will give us a new arbitrarily scaled translation

Back to visual odometry

We can find the pose of two cameras relative to each other

- How can we estimate the pose of a third camera?
- Using the essential matrix again will give us a new arbitrarily scaled translation
- Idea: Use the translation between the first two cameras to fix the scale.
- Triangulate points using the first two cameras
  - Use 3D points to find the pose of the third camera

Short break

Perspective-n-Point

Perspective-n-Point (PnP)

The Perspective-n-Point (PnP) problem.

Estimating the pose of a calibrated camera from n corresponding 3D-2D correspondences.

Kneip, Laurent, Davide Scaramuzza, and Roland Siegwart. "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation." CVPR 2011. IEEE, 2011.

PnP vs camera resectioning

In week 4 you did this for an uncalibrated camera (pest).

For an uncalibrated camera it is also called camera resectioning.

Naïve solution

- Estimate the projection matrix ($\tilde{\boldsymbol{P}}$)
- Compute $\boldsymbol{K}^{-1}\tilde{\boldsymbol{P}}$
- $\boldsymbol{K}^{-1}\tilde{\boldsymbol{P}} \approx \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix}$
  - $\boldsymbol{R}$ is likely not a proper rotation matrix
  - Requires many points

Perspective-n-Point (PnP)

- How many points are required?
- Ask yourself:
  - How many degrees of freedom does it have?
  - How many degrees of freedom does a single point fix?

Perspective-n-Point (PnP)

- How many points are required?
- Ask yourself:
  - How many degrees of freedom does it have?
  - How many degrees of freedom does a single point fix?
- Three correspondences are required
- This minimal case is therefore also known as P3P

P3P – Geometry

- The three 2D points give three pairwise angles
- The distances between the three 3D points give three pairwise distances

P3P – Geometry

- The three 2D points give three pairwise angles
- The distances between the three 3D points give three pairwise distances

From slides by Cyrill Stachniss

PnP rounding off

- Three correspondences generate four possible solutions, and a fourth correspondence can be used to choose the correct one
  - PnP requires 3+1 correspondences.
- Multiple algorithms exist and are implemented in OpenCV
- Use RANSAC to make it robust.

Pose vs. position

The pose of a camera is given by $\boldsymbol{R}$ and $\boldsymbol{t}$

This is not the orientation and position of the camera

$$\boldsymbol{T}_{\text{world}\to\text{cam}} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$$

$$\boldsymbol{T}_{\text{cam}\to\text{world}} = \boldsymbol{T}_{\text{world}\to\text{cam}}^{-1} = \begin{bmatrix} \boldsymbol{R}^\top & -\boldsymbol{R}^\top \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$$

Pose vs. position

The orientation of a camera is given by $\boldsymbol{R}^\mathsf{T}$

The position of a camera is given by $-\boldsymbol{R}^\mathsf{T} t$

This is important in order to plot the camera

Putting it all together

Outline

1. Use the essential or fundamental matrix to estimate the pose of the second camera
2. Triangulate points using the known camera poses
3. Use PnP or camera resectioning to estimate the pose of the next camera
4. Repeat steps 2. and 3.

Outline

1. Use the essential or fundamental matrix to estimate the pose of the second camera
2. Triangulate points using the known camera poses
3. Use PnP or camera resectioning to estimate the pose of the next camera
4. Repeat steps 2. and 3.
5. Use (windowed) bundle adjustment

Feature tracking

- Some points can be tracked through many frames
- Keep track of them to make your model drift less

Loop closure

The exercise

- Bigger exercise (two weeks)
- Estimate essential matrix
- Estimate 3D points
- SolvePnP

Learning objectives

After this lecture you should be able to:

- choose the correct decomposition of the essential matrix
- explain why the scale of the translation is unknown
- explain the Perspective-n-Point problem
- implement a simple visual odometry algorithm

Exercise time!

Structured light 3D scanning

Morten R. Hannemose, mohan@dtu.dk

May 9, 2023

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

This lecture is being livestreamed and recorded

(hopefully)

Learning objectives

After this lecture you should be able to:

- explain laser line scanning
- analyse and use Gray code encoding
- analyse and use phase shift encoding

Presentation topics

- Photogrammetry
- Structured light
  - Laser line scanning
  - Encoding surfaces
  - Gray code encoding
  - Phase shift encoding
- Notes on laser/projector calibration
- Exercise: structured light
- Exam information

Photogrammetry

Photogrammetry

Take a lot of pictures of a scene and use SLAM to find camera positions and 3D points. Algorithms for dense estimation of 3D points can be applied to get a full 3D scene.

Photogrammetry

Great:

- Works in daylight
- Handles textured objects

Bad:

- Requires good illumination
- Requires textured objects

Structured light

Laser line scanning

The first structured light technique.

- Laser projects 3D plane of light
- Camera sees the projected line
- Laser projector and camera are calibrated

Laser line scanning

Example laser line scanner triangulation:

1. detect laser line in image
2. triangulate 3D point as intersection of pixel and the laser plane
3. move laser line and goto 1

Robust method, with a few drawbacks

- Requires laser calibration.
- A slow method; one triangulation line per image

Encoding surfaces

Can we encode a surface?

Yes we can!

Encoding surfaces

Possibilities:

- Continuous encoding
  - Color or intensity gradient
  - Sinusoidal (phase) shifting
- Discrete encoding
  - Binary monochrome
  - ternary RGB encoding
  - quaternary CMYK encoding
- Other encoding schemes

Encoding surfaces

- Colored encodings have problems with colored surfaces
- Intensity codings have problems with textured objects
- Continuous encoding
  - **Sinusoidal (phase) shifting**
- Discrete encoding
  - **Binary monochrome**

Continuous encoding schemes

1. For each pixel in the camera, identify the code/color
2. For each code, identify the corresponding light plane
3. Triangulate using pixel rays and the "laser plane"

Can get a 3D point for each pixel in the camera

Not always robust.

Discrete encoding schemes

1. For each pixel in the camera, identify the code/color
2. For each code border, identify the corresponding plane
3. Triangulate using pixel rays and the "laser plane"

Only 3D points at code-borders, but usually more robust.

Binary encoding – single frame

Frame:

Inverted frame:

Frame + inverted frame makes us robust towards ambient light and varying object albedo.

Binary test: $\tau(\boldsymbol{p}, \boldsymbol{p}_i) = 1$ if $\boldsymbol{p} > \boldsymbol{p}_i$ else 0.

Single test; two regions; one border

Binary encoding – two frames

Frame 1:

Inverted frame 1:

Frame 2:

Inverted frame 2:

Two tests; four regions; three borders.

Binary encoding – multiple frames

Binary encoding

- For every frame we subdivide the 3D volume.
- For N frames we get 2^N unique regions.
- For N frames we get $2^N - 1$ unique borders.
- If a projector is $W = 1920$ pixels wide, we are limited to $N \leq \log_2(W) \approx 10.9$. That is a max of 10 frames in total or, 20 in total with inverted frames.

Binary encoding – border problems

Gray code encoding

Gray codes

- Only one bit flip at code borders; very robust.
- Uses same number of frames as binary patterns.

Phase shift encoding

Sinusoidal waves

$$I(x, y) = \frac{1}{2} + \frac{1}{2}\cos(n \cdot \theta(x))$$

Phase as a unique code

- The pattern is monochrome; good for colored objects.
- The phase is continuous; each point in space has a unique phase-plane.
- Even with a discrete intensity projector, the pattern is approximately continuous.
- $\theta(\cdot)$ is a function of the x-coordinate of the projector
  - From now on θ is the x-coordinate of the projector (from 0 to 2π)

Phase shifting

$$I(f,\theta) = \frac{1}{2} + \frac{1}{2}\cos\left(n \cdot \theta + 2\pi\frac{f}{s}\right)$$

When projecting sinusoids and shifting them, each pixel is a sinusoid as a function of the pattern

Phase shifting method

- Phase shift exactly one wavelength in s steps.
- The phase θ corresponds to a unique projector plane.
- We can find $n \cdot \theta$ for a single pixel by fitting a sinusoid to it

Phase shifting method

- Phase shift exactly one wavelength in s steps.
- The phase θ corresponds to a unique projector plane.
- We can find $n \cdot \theta$ for a single pixel by fitting a sinusoid to it
- This can be done with least squares (slow) or the fast Fourier transform (FFT) (fast).

$$\text{FFT}_f[I(f,\theta)] = \left\{\frac{s}{2}, \frac{s}{2}e^{i\theta}, 0, \ldots, 0\right\}$$

The second element of the FFT is a complex number with $\theta = \text{angle}(\text{FFT}_2)$.

Phase wrapping

The phase $n \cdot \theta = \text{angle}(\text{FFT}_2)$ is wrapped to $0 \leq n \cdot \theta \leq 2\pi$.

We need to unwrap the recovered phase!

Heterodyne principle

Use two sinusoids:

$$\begin{aligned}\theta_1 =& n_1 \cdot \theta \mod 2\pi \quad \text{(primary pattern)}\\ \theta_2 =& n_2 \cdot \theta \mod 2\pi \quad \text{(secondary pattern)}\end{aligned}$$

If we choose $n_2 = n_1 + 1$ we can recover θ by subtracting the secondary phase from the primary phase

$$\theta_2 - \theta_1 = n_2 \cdot \theta - n_1 \cdot \theta = \theta \mod 2\pi$$

This is the heterodyne principle, and gives us the phase cue (θ_c).

Heterodyne principle

Measured θ_1 and θ_2

$$\theta_c = \text{mod}\big(\theta_2(x) - \theta_1(x), 2\pi\big)$$

Looks great! 😀

Questions? Short break

Unwrapping

- If the measurements of θ_1 and θ_2 are noise free, the phase cue θ_c is exactly equal to θ.
- All measurements have noise, but in this case we can improve the handling of noise substantially.

Unwrapping

To make the system more robust to noise in the measurements we can compute θ using the phase cue and the primary phase θ_1.

The order counts how many times θ_1 has wrapped around

$$o_1 = \left\lfloor \frac{n_1 \cdot \theta_c - \theta_1}{2\pi} \right\rceil$$

The rounding $\lfloor \cdot \rceil$ makes it robust to noise. We can now estimate θ

$$\theta_{\text{est}} = \frac{2\pi o_1 + \theta_1}{n_1} \mod 2\pi$$

Unwrapping and phase cue – Motivation

- Why not just use the phase cue?
- $\theta_1 = \theta_1^{\text{true}} + \epsilon_1$
- $\theta_2 = \theta_2^{\text{true}} + \epsilon_2$
- $\theta_c = \theta^{\text{true}} - \epsilon_1 + \epsilon_2$

Unwrapping and phase cue – Motivation

- Why not just use the phase cue?
- $\theta_1 = \theta_1^{\text{true}} + \epsilon_1$
- $\theta_2 = \theta_2^{\text{true}} + \epsilon_2$
- $\theta_c = \theta^{\text{true}} - \epsilon_1 + \epsilon_2$
- The error of the phase cue is linear in the errors of θ_1 and θ_2
- θ has an error of $\frac{\epsilon_1}{n_1}$, which is much lower

Unwrapping – examples from today's exercise.

Examples from row 400 of camera 0.

Phase shift encoding

1. Project two sets of sinusoidals.
2. For each pixel, find the wrapped phases θ_1 and θ_2.
3. Calculate the phase cue $\theta_c = \theta_2 - \theta_1 \mod 2\pi$.
4. Calculate the primary order $o_1 = \left\lfloor \frac{n_1 \cdot \theta_c - \theta_1}{2\pi} \right\rceil$
5. Calculate the theta $\theta_{\text{est}} = \frac{2\pi o_1 + \theta_1}{n_1}$
6. For each phase, find the corresponding projector plane.
7. Triangulate pixel rays and the projector plane.

Phase shift encoding

- Very robust since the FFT is also a low-pass filter.
- Potentially one triangulation per camera pixel.
- Easy to make more precise; more projected frames gives a higher precision.

Notes on laser/projector calibration

Laser/projector calibration

- Is needed for many structured light methods.
- Requires correspondences between the laser/projector and the real world.
- Often we use a calibrated camera to relate projected lines/pixels to world coordinates.
- Not ideal for projectors, as they usually have poor quality lenses.

However, do we need to calibrate the laser/projector?

“Passive” structured light

- Use a stereo camera set up and a laser/projector.
- The laser/projector encodes the object surface, but is not calibrated.
  - Accurately calibrating a projector is harder than calibrating a camera
- Instead of laser/projector plane triangulation, we use epipolar lines in the cameras.
- Requires that the projected planes are not parallel with the epipolar planes.

Exercise: structured light

Structured light exercise

- Scan using phase shifting
- You will perform:
  - Image rectification
  - Phase decoding (and unwrapping)
  - Masking
  - Matching
  - Triangulation

Casper the baby T-Rex

Casper the baby T-Rex

Rectified cameras

Cameras are (virtually) made parallel to each other.

Masking

We need to only match phases, which is efficient and robust.

With rectified images, epipolar lines are the corresponding rows.

Live demo[ish]

Further courses

- Are you interested in working more with images?
- 02506 Advanced Image Analysis (Spring)
- 02514 Deep Learning in Computer Vision (June)
- Advanced topics in deep learning in computer vision (Fall23, then every Spring)

Thesis projects

Are you interested in doing your thesis (bachelors or masters) within computer vision or deep learning?

Contact me! Let's find out if I we can find a project that fits your interests and skills.

Learning objectives

After this lecture you should be able to:

- explain laser line scanning
- analyse and use Gray code encoding
- analyse and use phase shift encoding

Exam information

Exam information

- Examples of exam questions have been uploaded.
- Multiple choice exam
  - No negative points for wrong answer
- Questions are a mix of understanding and calculation in Python.
  - No internet access during exam
  - You can download the documentation for OpenCV in advance

Exercise time!