C18 Computer Vision. This time ...

1. Introduction; imaging geometry; camera calibration.
2. Salient feature detection edges, line and corners.
3. Recovering 3D from two images I: epipolar geometry.
4. Recovering 3D from two images II: stereo correspondence algorithms; triangulation.
5. Structure from Motion I: computing the fundamental matrix by matching corners, robust matching with RANSAC.
6. Structure from Motion II: SIFT, recovering egomotion.
7. Image motion: optic flow; the aperture problem; visual tracking.
8. Object recognition and detection.

Outline

Introduction
Computing $H$ or $F$ from point matches
Feature detection and matching
RANSAC
SIFT
Determining the ego-motion from $F$
Structure and motion from more than two views
Introduction

In the previous lectures we have seen stereo reconstruction from two views:

1. Obtain (somehow) the camera parameters $P = K[I|0]$ and $P' = K'[R|t]$.
2. Compute the fundamental matrix $F = K'^{-T} [t] 	imes RK^{-1}$
3. Match points $x$ in an image to corresponding points $x'$ in the second along the epipolar lines $l' = Fx$.
4. Triangulation: compute the 3D points $X$ from $x, x', P, P'$.

Next – What happens if:

1. you do not know how the camera parameters?
2. you have more than two images?

You get Structure from Motion

[Carl Olsson]

The Structure from Motion (SFM) problem

Given two or more images of a scene:

1. Match corner points to find point correspondence. This is harder than before as the epipolar geometry is unavailable.
2. Compute the egomotion $R, t$:
   - For planar scenes:
     - Compute the homography matrix $H$ (e.g. four points algorithm seen in B14);
     - Extract the egomotion from $H$.
   - For general 3D scenes:
     - Compute the fundamental matrix $F$ (e.g. eight points algorithm);
     - Extract the egomotion from $F$.
3. Triangulate as before to obtain the 3D points.

Prototypical SFM pipeline

Assumptions:

- **Known** intrinsic calibration $K, K'$.
- **Unknown** extrinsic calibration $R, t$ (egomotion).
Corner points computed for each frame

Extract some corner points, for example using the Harris detector:

Egomotion from corner points

Egomotion = transformation between the cameras.

- Given point correspondences $x_i \leftrightarrow x'_i$ for $i = 1 \ldots n$, we want to determine $R$ and $t$.
- Intuition: Keep $C$ still, and move $C'$ until all rays intersect.
- Obviously three correspondences are not enough to fix $C'$. How many do we need?
Outline of egomotion computation

1. Compute the fundamental matrix $F$ from the correspondences $x_i \leftrightarrow x'_i$.
2. Decompose $F = K^{-1} [t] \times RK^{-1}$ to find $R$, $t$ (given the known $K$ and $K'$).
3. Compute the projection matrices $P$ and $P'$ if needed.

Actually, $t$ found only up to scale

$F$ is a homogeneous matrix, so 

$$F \propto K^{-1} [t] \times RK^{-1} \propto K^{-1} [\lambda t] \times RK^{-1}$$

Therefore translation and all lengths are recovered only up to scale:

$$t \equiv \lambda t, \quad X_i \equiv \lambda X_i.$$ 

Depth/scale ambiguity

We cannot distinguish:

- a large translation when viewing a large distant scene; from
- a small translation when viewing a small near-to scene.

Question: How might you resolve the depth/scale scaling ambiguity?

How many correspondences are required?

Because of the depth/speed scaling ambiguity the rotation (3 DoF) can be determined completely but only the translation direction (2 DoF) is recoverable.

This allows us to evaluate the number of correspondence needed:

1. For $n$ scene points there are $3n$ unknowns
2. Between 2 views there are $5 = (3 \text{ rot} + 2 \text{ trans})$ unknowns
3. Each correspondence yields 4 measurements
4. Hence $4n \geq 3n + 5$ and

$$n \geq 5 \text{ correspondences are needed}$$

For $n < 7$ the solutions are non-linear, so we’ll see solutions for $n = 7$ and $n = 8$. 

Computing the fundamental matrix for $n \geq 8$

Task: Given $n$ correspondences $x_i \leftrightarrow x'_i$ compute $F$ such that

$$\forall i : x'_i^T F x_i = 0.$$ 

Solution: Each correspondence generates one constraint

$$\begin{bmatrix} x'_i & y'_i & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

which can be written as

$$x'_i x_i f_1 + x'_i y_i f_2 + x'_i f_5 + y'_i x_i f_4 + y'_i y_i f_5 + y'_i f_6 + x_i f_7 + y_i f_8 + f_9 = 0$$

or

$$\begin{bmatrix} x'_i x_i & x'_i y_i & x'_i & y'_i x_i & y'_i y_i & x_i & y_i & 1 \\ f_1 & : & f_9 \end{bmatrix} = 0.$$
Computing the fundamental matrix /ctd

▶ For $n$ correspondences build up the $n \times 9$ system

$$A_{n \times 9}f = \begin{bmatrix} x'_1 x_1 & x'_1 y_1 & x'_1 & y'_1 x_1 & y'_1 & y'_1 y_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_n x_n & x'_n y_n & x'_n & y'_n x_n & y'_n & y'_n y_n & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}.$$  

▶ For $n = 8$ points $f$ can be found as the null-space of $A$, and so $f$ and $F$ are determined up to scale (as expected).

▶ Since the points are noisy, in general one wants to use $n > 8$. This can be done using least square.

Proof of the eigendecomposition solution

▶ The squared sum of the residuals $r = Af$ is

$$\|r\|^2 = r^\top r = f^\top A^\top Af = f^\top Mf$$

▶ $M = A^\top A$ is a $n \times n$ symmetric real matrix; hence it can be decomposed as

$$M = \mathbf{V} \Lambda \mathbf{V}^\top = \mathbf{V} \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \\ & & & \lambda_n \end{bmatrix} \mathbf{V}^\top = \sum_{i=1}^n \lambda_i \hat{e}_i \hat{e}_i^\top$$

where

▶ $\mathbf{V} = [\hat{e}_1 \ldots \hat{e}_n]$ is the orthonormal matrix of eigenvectors

▶ eigenvalues are non-decreasing: $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.

▶ The eigenvalues are non-negative because:

$$\hat{M} \hat{e}_i = \hat{e}_i \lambda_i \Rightarrow \hat{e}_i^\top \hat{M} \hat{e}_i = \hat{e}_i^\top \hat{e}_i \lambda_i \Rightarrow \| \mathbf{A} \hat{e}_i \|^2 = \lambda_i \geq 0.$$  

Then

$$f^\top Mf = \lambda_1 (f^\top \hat{e}_1)^2 + \lambda_2 (f^\top \hat{e}_2)^2 + \ldots + \lambda_n (f^\top \hat{e}_n)^2.$$  

▶ This expression is minimised when $f = \hat{e}_1$.

Proof of the SVD solution

▶ Any $m \times n$ matrix $A$ where $m \geq n$ can be decomposed as

$$A_{m \times n} = \mathbf{U}_{m \times n} \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \\ & & & \sigma_n \end{bmatrix}_{n \times n} \mathbf{V}_{n \times n}^\top$$

where $\mathbf{U}$ is column-orthogonal, $\mathbf{V}$ is fully orthogonal, and $\Sigma$ contains the singular values ordered so $0 \leq \sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_n$.

▶ The singular vectors $\mathbf{V}$ of $A$ are the same as the eigenvectors of $M = A^\top A$:

$$M = A^\top A = \mathbf{V} \Sigma^\top \mathbf{U}^\top \mathbf{U} \Sigma \mathbf{V}^\top = \mathbf{V} \Sigma^2 \mathbf{V}^\top$$

▶ In particular $f = \hat{e}_1$ is the first column of $\mathbf{V}$.

▶ The SVD is usually preferred to the eigenvalue decomposition because it is numerically more stable.

A least squares version of the 8-point algorithm

Due to noise, there will not be an exact solution to $Af = 0$ ($A$ has full rank).

Least square formulation

Find the unit vector $f$ that minimizes the norm of the residual $r = Af$:

$$f^* = \arg\min_{f \|f\|=1} \|Af\|^2$$

Solution with eigenvalues

Compute the eigen-decomposition of the matrix $M = A^\top A$ and set $f$ to the (unit) eigenvector $\hat{e}_1$ corresponding to the smallest eigenvalue $\lambda_1$.

Solution with SVD

Compute the SVD of the matrix $A$ and set $f$ to the (unit) right singular vector $\hat{e}_1$ corresponding to the smallest singular value $\sigma_1$.  

Proof of the SVD solution

▶ Any $m \times n$ matrix $A$ where $m \geq n$ can be decomposed as

$$A_{m \times n} = \mathbf{U}_{m \times n} \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \\ & & & \sigma_n \end{bmatrix}_{n \times n} \mathbf{V}_{n \times n}^\top$$

where $\mathbf{U}$ is column-orthogonal, $\mathbf{V}$ is fully orthogonal, and $\Sigma$ contains the singular values ordered so $0 \leq \sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_n$.

▶ The singular vectors $\mathbf{V}$ of $A$ are the same as the eigenvectors of $M = A^\top A$:

$$M = A^\top A = \mathbf{V} \Sigma^\top \mathbf{U}^\top \mathbf{U} \Sigma \mathbf{V}^\top = \mathbf{V} \Sigma^2 \mathbf{V}^\top$$

▶ In particular $f = \hat{e}_1$ is the first column of $\mathbf{V}$.

▶ The SVD is usually preferred to the eigenvalue decomposition because it is numerically more stable.
Computing $F$ from 7 points

- For the $7 \times 9$ set of equations $A f = 0$ we know that $f$ is in the null space of $A$.
- This null space is 2-dimensional and hence spanned by two vectors $f_1$ and $f_2$. Since $f$ is determined up to scale, all solutions are given by:

$$f = \alpha f_1 + (1 - \alpha) f_2$$

- Reshaping the vectors, results in a family of candidate fundamental matrices

$$F = \alpha F_1 + (1 - \alpha) F_2$$

- To find which one is a “proper” fundamental matrix, use the non-linear constraint $\det F = 0$. This gives a cubic equation in $\alpha$. Can you see why?
- The cubic has either one or three real solutions for $\alpha$.

A visual compass

Use $H_\infty$ to register images to a common reference frame to create panoramic mosaic:

A Visual Compass

If the motion of the camera is known to be a pure rotation, then the images are related by an homography

$$x' = H_\infty x$$

where

$$H_\infty = K' RK^{-1}.$$  

Algorithm:

- Find correspondences $x_i \leftrightarrow x_i'$
- Compute $H_\infty$ from the correspondences (see B14)
- Extract $R$ to find relative rotations

But of course we cannot recover any scene structure!

Outline

- Introduction
- Computing $H$ or $F$ from point matches
- Feature detection and matching
  - RANSAC
  - SIFT
- Determining the ego-motion from $F$
- Structure and motion from more than two views
Feature detection, matching and the $F$ matrix

So far, we have not discussed matching. The reason is that computation of the fundamental matrix can be incorporated into the matching.

Outline:

- Extract image points as corners. Why corners?
- Obtain an initial corner matches using local descriptors.
- Remove outlier and estimate the fundamental matrix $F$ using RANSAC.
- Obtain further corner matches using $F$.

Why corner points, especially?

- Why not use lines, or take a dense pixel-based approach?
- The key reason is that the search for matches is no longer 1D when the camera motion is unknown. A 2D region has to be searched.
- A dense approach is then likely to be too expensive, and matching sections of a line suffers the aperture problem.
- Corners are:
  - relatively sparse;
  - reasonably cheap to compute;
  - well-localized;
  - appear quite robustly from frame to frame.
- Hence corners are good for matching.

Corner Points computed for each frame

Recall that points with distinctively high autocorrelation provide the best chance of deriving a distinctive cross-correlation signal.

Initial matching

- Extract corners in both images (feature detection).
- For each corner $x$ in $C$, make a list of potential matches $x'$ in a region in $C'$ around $x$ (heuristic).
- Rank the matches by comparing the regions around the corners using cross-correlation.
- Sift them to reconcile forward-backward inconsistencies.
- The idea here is to not do too much work — just enough to get some good matches.
Matches — some good matches, some mismatches. Can still compute $F$ with around 50% mismatches. How?

RANSAC – RANdom SAmple Concensus

- Suppose you tried to fit a straight line to data containing outliers — points which are not properly described by the assumed probability distribution.
- The usual methods of least squares are hopelessly corrupted.
- Need to detect outliers and exclude them.

- Use estimation based on robust statistics RANSAC was the first, devised by vision researchers, Fischler & Bolles (1981)
RANSAC algorithm for lines

1. For many repeated trials:
   1.1 Select a random sample of two points
   1.2 Fit a line through them
   1.3 Count how many other points are within a threshold distance of the line (inliers)
2. Select the line with the largest number of inliers
3. Refine the line by fitting it to all the inliers (using least squares)

Remarks:
▶ Sample a minimal set of points for your problem (2 for lines).
▶ Repeat such that there is a high chance that at least one minimal set contains only inliers (see tutorial sheet).

RANSAC algorithm for F

1. For many repeated trials:
   1.1 Select a random sample of seven correspondences
   1.2 Compute F using the cubic method
   1.3 Count how many other correspondences are within threshold distance of the epipolar lines (inliers)
2. Select the F with the largest number of inliers
3. Refine F by fitting it to all the inliers (using the SVD method)
RANSAC algorithm for $H$

1. For many repeated trials:
   1.1 Select a random sample of four correspondences
   1.2 Compute $H$ (as in B14)
   1.3 Count how many other correspondences are within threshold distance of the predicted locations (inliers)
2. Select the $H$ with the largest number of inliers
3. Refine $H$ by fitting it to all the inliers, optimizing the reprojection error

$$
\min_H \sum_{(x, x') \in \text{Inliers}} d^2(x', Hx) + d^2(H^{-1}x', x)
$$

---

Epipolar geometry

---

C18 Computer Vision. This time ...

1. Introduction; imaging geometry; camera calibration.
2. Salient feature detection edges, line and corners.
3. Recovering 3D from two images I: epipolar geometry.
4. Recovering 3D from two images II: stereo correspondence algorithms; triangulation.
5. Structure from Motion I: computing the fundamental matrix by matching corners, robust matching with RANSAC.
6. Structure from Motion II: SIFT, recovering egomotion.
7. Image motion: optic flow; the aperture problem; visual tracking.
8. Object recognition and detection.
Co-variant detection with SIFT

**Co-variant feature detection** means that the same “physical features” are extracted regardless of viewpoint-induced image transformations:

- Intuition: the extracted features track (“co-vary”) with the image transformation.
- Hence it is meaningful to match them.

SIFT achieves co-variant detection by selecting **blob-like structures**:

The implicit assumption is that blobs will still look like blobs after the image is transformed.

Blob detection

SIFT detects blobs as local maxima in the response of a blob-like image filter. The filter is the (scaled) **Laplacian of Gaussian**:

\[ \sigma^2 \nabla^2 G(x, y, \sigma) \]

The LoG filter is convolved with the image to obtain a blob response or LoG:

\[ \text{LoG}(x, y, \sigma) = \sigma^2 \nabla^2 G(x, y, \sigma) \ast I(x, y), \quad G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\left(x^2 + y^2\right)/2\sigma^2} \]
Multi-scale blob response

Convolution searches for blobs at all locations \((x, y)\). We also need to search at all scales \(\sigma\).

Repeat for \(\sigma(o, s) = \sigma_0 2^{(o + \frac{s}{2})}\) where \(o = \) octave and \(s = \) octave subdivision:

First octave \(o = 0\):

\[
\begin{array}{cccc}
\sigma_0 2^{-\frac{1}{2}} & \sigma_0 2^0 & \sigma_0 2^{\frac{1}{2}} & \sigma_0 2^{\frac{3}{2}}
\end{array}
\]

Next octave \(o = 1\):

\[
\begin{array}{cccc}
\sigma_0 2^{1-\frac{1}{2}} & \sigma_0 2 & \sigma_0 2^{1+\frac{1}{2}} & \sigma_0 2^{1+\frac{3}{2}}
\end{array}
\]

Next octave \(o = 2\):

\[
\begin{array}{cccc}
\sigma_0 2^{2-\frac{1}{2}} & \sigma_0 2^{2} & \sigma_0 2^{2+\frac{1}{2}} & \sigma_0 2^{2+\frac{3}{2}}
\end{array}
\]

\[
\ldots
\]

Multi-scale blob localisation

So far we have computed the LoG scale space:

\[
\text{LoG}(x, y, \sigma) = \sigma^2 \nabla^2 G(x, y, \sigma) \ast I(x, y).
\]

Blobs = local maxima and minima of the LoG scale space:

Computation: for each value \(\text{LoG}(x, y, \sigma)\), check the \(3 \times 3 \times 3\) neighbours in scale and space.

If the value is larger (or smaller) than all the neighbours, mark \((x, y, \sigma)\) as a blob.

Normalisation in the LoG

The \(\sigma^2\) factor in the LoG definition makes the different scales comparable so that computing local maxima in scale is meaningful.

Gaussian vs Laplacian scale spaces

For efficiency, it is often better to start from the Gaussian scale space

\[
L(x, y, \sigma) = G(x, y, \sigma) \ast I(x, y)
\]

instead of the LoG scale space

\[
\text{LoG}(x, y, \sigma) = \sigma^2 \nabla^2 G(x, y, \sigma) \ast I(x, y).
\]

Difference of Gaussians (DoG)

The DoG scale space is the difference of consecutive levels of the Gaussian scale space:

\[
\text{DoG}(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \ast I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma).
\]

where \(k > 1\) is a small multiplicative factor. Then the LoG is given by:

\[
\text{LoG}(x, y, \sigma) \approx \frac{1}{k - 1} \left( L(x, y, k\sigma) - L(x, y, \sigma) \right).
\]
Scale space pyramid

After each octave, the sampling resolution can be halved to avoid storing and processing redundant data:

Scale space pyramid computation

1. Blur (convolve with a Gaussian kernel) to go to the next scale level.
2. Down-sample by half to go to the next scale octave.
3. Subtract pairs to go from Gaussian scale space to DoG.

DoG ≈ LoG scale space

DoG ≈ LoG scale space (pyramid)
Rotational invariance

- Create histogram of local gradient directions computed at selected scale
- Assign canonical orientation at peak of smoothed histogram
- Each key specifies stable coordinates $x$, $y$, scale, orientation

Feature descriptor

Now use local image data (at the intrinsic scale) to create a **local feature descriptor**

- Thresholded image gradients are sampled over $16\times16$ array of locations in scale space
- Create array of orientation histograms
- 8 orientations $\times 4\times4$ histogram array $= 128$ dimensions

Examples

- Demo?
- Autostitch
Computing \( R \) and \( t \) from \( F \)

Recall that \( F = K'^{-1}[t] \times RK^{-1} \). We now show how to recover \( R \) and \( t \) from \( F \) (given \( K \) and \( K' \)).

1. Compute the essential matrix \( E = [t] \times R = K'^T FK \).
2. Compute \( t \) as the null vector of \( E^T \) (i.e. \( E^T t = 0 \)).
   - \( t \) is determined up to a scaling factor \( \mu \).
   - there are two solutions \( \pm \mu t \)
3. Compute \( R \) from \( E \)
   - the algorithm for this step is given later.
   - it returns two solutions \( R_1 \) and \( R_2 \).
4. Overall, there are four solutions for the projection matrix:
   \[
   P' = K'[R_1| \mu t] \\
   P' = K'[R_1| - \mu t] \\
   P' = K'[R_2| \mu t] \\
   P' = K'[R_2| - \mu t]
   \]
5. Exclude 3 of these using a visibility test

### The four solutions

The 3D point is in front of both cameras in only one case.

Note these are “computer vision” cameras, so to be visible a ray must pass through the image on its way to the optic centre!
Structure and motion for more than two views

Why bother?

1. Matching becomes more verifiable, as 3D point estimates are available to reproject.
2. 3D point estimates improve as further views over a range of angles is obtained.
3. There is no increase in the degree of ambiguity, though the overall scale ambiguity persists.

Notation for three+ views

For three views let the cameras be $C, C', C''$ with projection matrices $P, P'$ and $P''$, and with image points $x, x'$ and $x''$.

Point correspondence over 3 views

Given the projection matrices and $x \leftrightarrow x'$ how is the point $x''$ found?

Algorithm:

1. Compute the 3D point from $x$ and $x'$
2. Then re-project using $P''$.
3. The search in the third image is zero-D, and the size of the search region depends only on uncertainty.
Problem statement: structure and motion

Given: \( n \) matching image points \( x_i^j \) over \( m \) views

Find: the cameras \( P_i \) and the 3D points \( X_j \) such that \( x_i^j \approx P_i X_j \) by finding:

\[
\min_{P_i, X_j} \sum_{j=1}^n \sum_{i=1}^m d^2(x_i^j, P_i X_j)
\]

This is a serious minimization:

- For each camera, 6 parameters
- For each 3D point, 3 parameters
- Total of \( 6m + 3n - 1 \) (–1 for scale) parameters overall

For 50 frames, 1000 points, we have \( 3.3 \times 10^3 \) unknowns!

Building block is computing correspondences \( x_i^j \leftrightarrow x_i^{j+1} \), finding \( F^{i+1}_i \) and then matrices \( P_i, P_i^{i+1} \)

Algorithm

1. Compute interest points in each image
2. Compute matches between consecutive image pairs \( i, i + 1 \)
3. Compute \( F^{i+1}_i \). Recover \( P_i, P_i^{i+1} \)
4. Compute scene points
5. Extend correspondences over image triples
6. Extend correspondences over all images
7. Optimize over all \( P_i, X_j \)

2d3’s Boujou system
Zisserman, Fitzgibbon, Torr, Beardsley

Original Sequence

2d3’s Boujou system
Zisserman, Fitzgibbon, Torr, Beardsley

Augmentation
Batch SFM

- Up to now **batch, offline** processing of video sequences
- Post-production, 3D model reconstruction, etc.

Real-time, sequential SFM

- **Real-time, sequential**, fixed time budget (10s of milliseconds)
- Build and maintain a map, and localise w.r.t. the map
- Real-time robotics applications, but in simplified 2D environments, specialised sensors, etc
- Reliable, repeated measurement is crucial – mitigates against drift giving **repeatable accuracy**.

Sequential structure from motion: visual SLAM

- Represent joint distribution over camera and feature positions using a single **multi-variate Gaussian**.
  \[
  \mathbf{x} = \begin{bmatrix} x_v \\ y_1 \\ y_2 \\ \vdots \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} P_{xx} & P_{x y_1} & P_{x y_2} & \cdots \\
  P_{y_1 x} & P_{y_1 y_1} & P_{y_1 y_2} & \cdots \\
  P_{y_2 x} & P_{y_2 y_1} & P_{y_2 y_2} & \cdots \\
  \vdots & \vdots & \vdots & \ddots \end{bmatrix}
  \]
- Use Kalman Filter (see C4B Mobile Robotics)
  - **predict** → **measure** → **update**
  - framework to propagate uncertainty, and fuse measurement data

Example: real-time, sequential structure from motion
Real-Time Camera Tracking in Unknown Scenes

Davison, Reid, Smith, Williams, Klein, et al.