VISUAL INERTIAL ODOMETRY (VO)

- **WHY?**
  - Very accurate: relative position error is 0.1% - 2% travelled distance
  - Not affected by wheel slippage
  - Complementary to
    - Wheel encoders (wheel odometry)
    - GPS (when degraded)
    - IMUs
    - Laser

- **ASSUMPTIONS**
  - SUFFICIENT ILLUMINATION in the environment
  - DOMINANCE OF STATIC SCENE over moving objects
  - ENOUGH TEXTURE to allow apparent motion to be extracted
  - Sufficient SCENE OVERLAP between consecutive frames

STRUCTURE FROM MOTION (SFM)

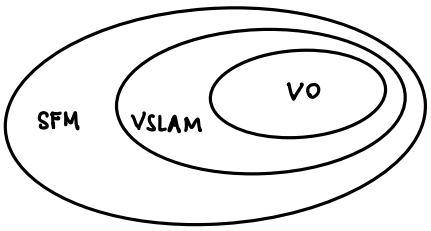
SFM is more general than VO and tackles the problem of 3D RECONSTRUCTION AND 6DOF POSE ESTIMATION from UNORDERED IMAGE SET

- VO is a particular case of SFM.
- VO focuses on estimating the 6DOF motion of the camera SEQUENTIALLY (as a new frame arrives) and in REAL TIME.
- Sometimes SFM is used as synonim of VO

VISUAL SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

SLAM = VO + LOOP DETECTION & CLOSURE

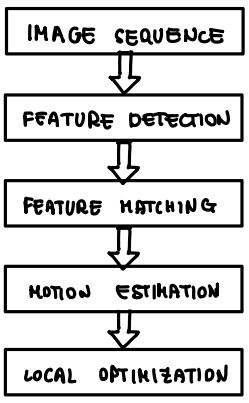
It guarantees global consistency: the estimated trajectory is globally correct from start to end



TAXONOMY OF THE METHODS

VO FLOW CHART

VO computes the camera path incrementally (pose after pose)



(Tracking)

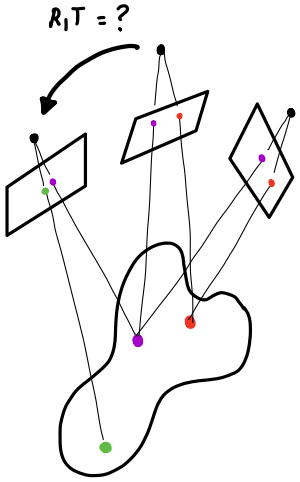
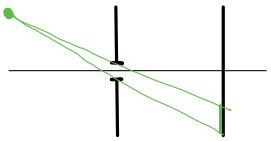


IMAGE FORMATION

THE PINHOLE CAMERA

In an **IDEAL PINHOLE**, only one ray of light reaches each point on the film



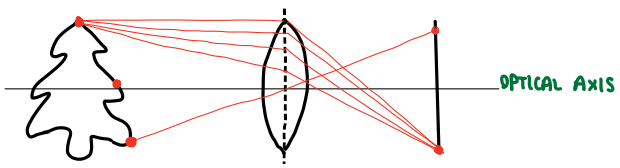
Shrinking the aperture helps but

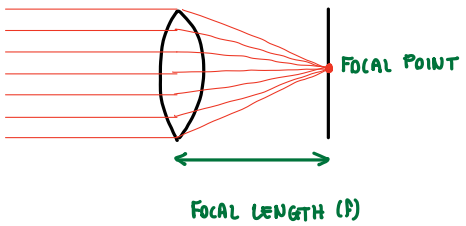
- **LESS LIGHT** gets through (must increase the exposure)
- **DIFFRACTION EFFECTS** arise

USING A CONVERGING LENS

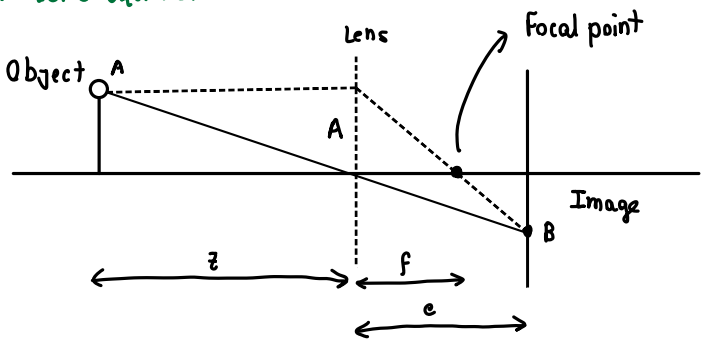
We can use a lens that focuses light onto the film

- ray passing through the **OPTICAL CENTER** are not deviated
- All rays parallel to the optical axis converge to the **FOCAL POINT**





THIN LENS EQUATION



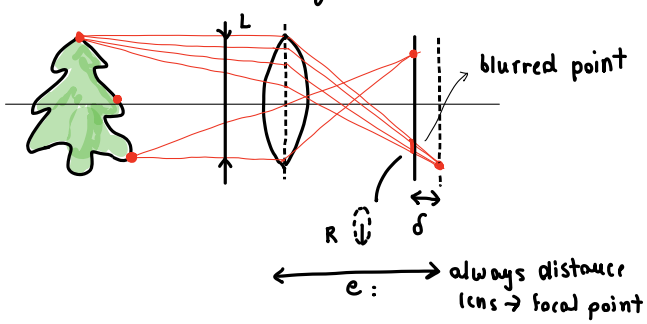
$$\frac{B}{e} = \frac{A}{z} \rightarrow \frac{B}{A} = \frac{e}{z}$$

$$\frac{B}{A} = \frac{e-f}{f} = \frac{e}{f} - 1$$

$$\frac{e}{f} - 1 = \frac{e}{z} \Rightarrow /e \quad \frac{1}{f} - \frac{1}{e} = \frac{1}{z}$$

$$\boxed{\frac{1}{f} = \frac{1}{z} + \frac{1}{e}}$$

- Any object satisfying this equation is **IN FOCUS**
- Can we use this to measure distances?
- For a **FIXED FILM DISTANCE** e, there is a specific distance between the object and the lens (z) at which the object appears in "focus"
- Other points project to a "blur circle" in the image

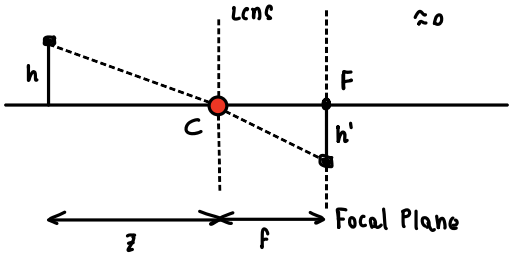


$R = L\delta / 2e$

- A small pinhole (L) gives a small blur (R)
- To capture a "good" image: adjust camera settings

THE PINHOLE APPROXIMATION

For distant objects $z \gg L$, $z \gg f$: $\frac{1}{f} = \frac{1}{z} + \frac{1}{e} \Rightarrow \frac{1}{f} \approx \frac{1}{e} \Rightarrow f \approx e$



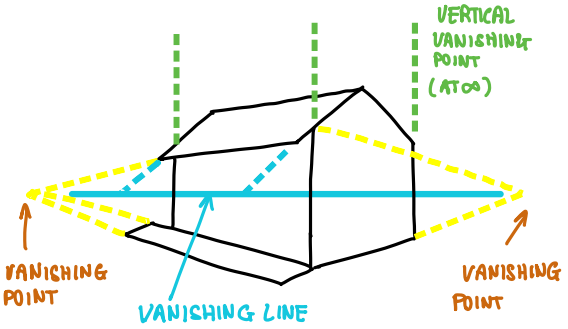
C: Optical center or Center of Projection $\Rightarrow \frac{h'}{h} = \frac{f}{z} \Rightarrow$ $\boxed{h' = \frac{f}{z} h}$

- The dependence of the image of an object on its depth (distance from camera) is known as PERSPECTIVE. (e.g. far objects appear small)

PERSPECTIVE

- straight lines are still straight
- length is lost
- angles are lost

} parallelism & perpendicularity ✗



VANISHING POINTS AND LINES

- Parallel lines in the world intersect in the image at a VANISHING POINT
- Parallel planes in the world intersect in the image at a VANISHING LINE

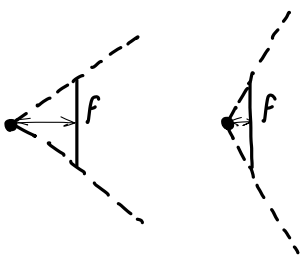
FOCUS AND DEPTH OF FIELD (DoF)

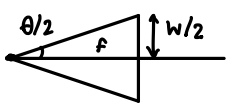
- The DEPTH OF FIELD (DoF) is the distance between the nearest and farthest object in the scene that appear acceptably sharp in an image
- Although sharpness is "perfect" at a single distance, in the DoF loss of sharpness is imperceptible
- A smaller aperture increases the DoF but reduces the amount of light into the camera

FIELD OF VIEW (FOV)

Angular measure of portion of 3D space seen by the camera

- As f gets smaller, image becomes more wide angle → can see more points
- As f gets larger, image becomes more narrow angle → can see less points



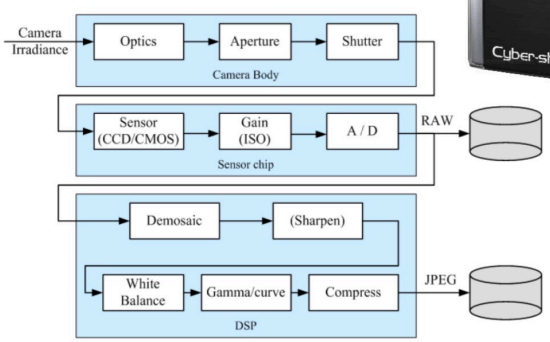


$\boxed{\tan\left(\frac{\theta}{2}\right) = \frac{w}{2f}}$

DIGITAL CAMERAS

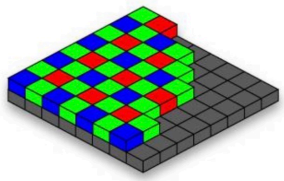
The film is an array of CCD of CMOS light sensitive devices that convert photons into electrons

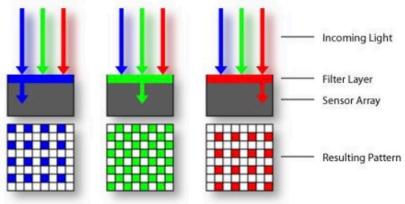
- Pixel intensity with 8 bits ranges between [0, 255]



COLOR SENSING IN DIGITAL CAMERAS

Bayer grid





- The BAYER PATTERN:
  - 1/2 filter is GREEN → humans are much more sensitive to green
  - 1/4 filter is RED
  - 1/4 filter is BLUE
- For each pixel we estimate missing color component from neighboring values (DEMOSAICING)

TYPES OF SHUTTER

ROLLING SHUTTER:
- Rows of pixels are exposed at different times, one after the other
- Distort image for moving objects

GLOBAL SHUTTER:
- All pixels are exposed simultaneously
- Good for moving objects → no distortion

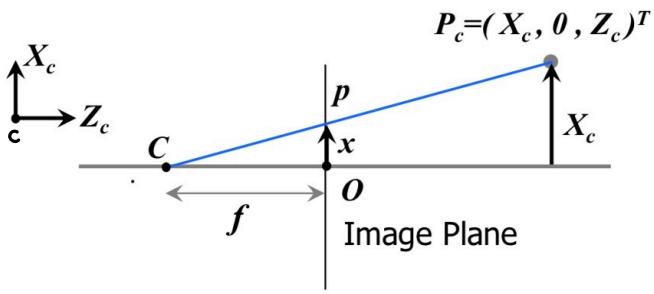
PERSPECTIVE CAMERA

Goal: Find pixel coordinates (u,v) of point P_w in the world frame:

1) Convert P_w to camera point P_c through extrinsics (transform $[R|t]$)
2) Convert P_c to image plane coordinates (x,y)
3) Convert (x,y) to discretized pixel coordinates (u,v)

1) $P_c = [R|t]\,P_w$ P is in homogeneous coordinates and $[R|t] = \begin{bmatrix} R & t \end{bmatrix} \in \mathbb{R}^{3\times 4}$

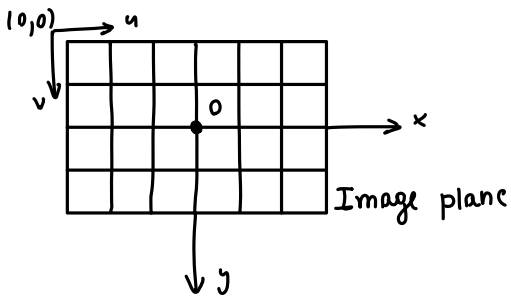
2)



$$\frac{x}{f} = \frac{X_c}{Z_c} \Rightarrow x = f\frac{X_c}{Z_c}$$

Similarly $\Rightarrow y = f\frac{Y_c}{Z_c}$

3) Convert (x,y) to (u,v)



- pixel coords of O are (u_0, v_0)
- Scale factors k_u, k_v for the pixel size in both dimensions

$$\begin{aligned} u &= u_0 + k_u x \\ v &= v_0 + k_v y \end{aligned}$$

- Use HOMOGENEOUS COORDINATES for linear mapping from 3D to 2D, by introducing an extra element (scale)

$$p = \begin{pmatrix} u \\ v \end{pmatrix} \Rightarrow \tilde{p} = \begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

- It follows that

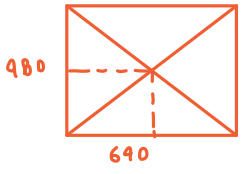
$$\begin{aligned} u &= u_0 + k_u f X_c / Z_c \\ v &= v_0 + k_v f Y_c / Z_c \end{aligned} \Rightarrow \begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

CALIBRATION MATRIX

EXERCISE 1

- Determine K of a camera 640×480 pixels with FOV = 90°
- The principal point is in the center of the diagonals
- Also, pixels are square
- What is the vertical field of view?

$\tan\left(\frac{\theta}{2}\right) = \frac{W}{2f} \Rightarrow \tan(45°) = \frac{640}{2f} \Rightarrow f = 320$ pixels



$u_0 = 320$
$v_0 = 240$

Square pixels → $k_u = k_v = 1$ ⇨ $K = \begin{bmatrix} 320 & 0 & 320 \\ 0 & 320 & 240 \\ 0 & 0 & 1 \end{bmatrix}$

$$\theta_v = 2\arctan\left(\frac{W}{2f}\right) = 2\arctan\left(\frac{480}{2 \cdot 320}\right) = 2\arctan\left(\frac{3}{4}\right) = 73.74°$$

EXERCISE 2

- Prove that world parallel lines intersect at a vanishing point in the camera image

Equation of a 3D line: $\vec{p_0} + \vec{d}t$, $t \in \mathbb{R}$ $\quad \vec{d} = [l \; m \; n]$

$$\lim_{t \to \infty} x_v = \lim_{t \to \infty} \frac{f(x_0 + tl)}{(z_0 + tn)} = \frac{fl}{n}$$

$$\lim_{t \to \infty} y_v = \lim_{t \to \infty} \frac{f(y_0 + tm)}{(z_0 + tn)} = \frac{fm}{n}$$

PERSPECTIVE CAMERA (CONTINUED)

From the real world to the camera image $\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = [R \mid t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$

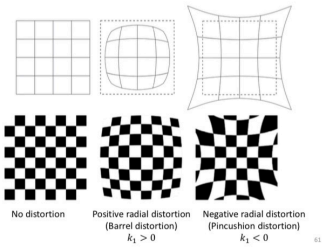
PERSPECTIVE PROJECTION EQUATION $\boxed{\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R \mid t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}}$

We define the NORMALIZED IMAGE COORDINATES as $\begin{bmatrix} \bar{u} \\ \bar{v} \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$ (as the focal length were 1)

LENS DISTORTION

RADIAL DISTORTION

- For a given non distorted image point (u,v) the amount of distortion is non linear of its distance r from the principal point
- A simple quadratic model often works



$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = (1+k_1 r^2)\begin{bmatrix} u-u_0 \\ v-v_0 \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}$$

$$r^2 = (u-u_0)^2 + (v-v_0)^2$$

- Depending on the distortion's amount (and thus the camera FOV), higher order terms can be introduced

TANGENTIAL DISTORTION

- If the lens is misaligned (not perfectly orthogonal to the image sensor) a non-radial distortion appears.

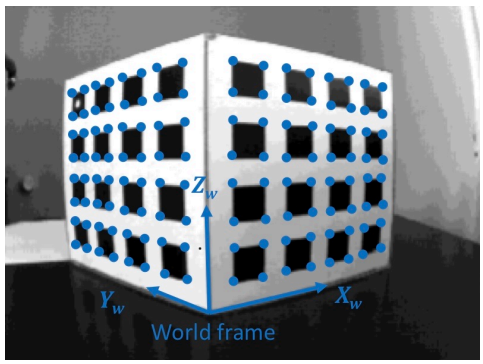
CAMERA CALIBRATION

CALIBRATION: the process to determine the intrinsic K and extrinsic (R,T).
(optionally also the lens distortion)
There are 2 popular methods

- **TSAI'S METHOD** : uses 3D objects
- **ZHANG'S METHOD** : uses planar grids

TSAIS'S METHOD

- measure the 3D position of $n \geq 6$ control points on a 3D calibration target and their 2D coordinates of their projection in the image.



- The method of Direct Linear Transform (DLT) rewrites the perspective projection as a HOMOGENEOUS LINEAR EQUATION and solves it by standard methods.

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = k[R|t] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & & & \\ \vdots & & \ddots & \\ & & & m_{34} \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}}_{P} = \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$\rightarrow \quad u = \frac{\lambda u}{\lambda} = \frac{m_1^T P}{m_3^T P} \quad v = \frac{\lambda v}{\lambda} = \frac{m_2^T P}{m_3^T P} \quad \rightarrow \quad (m_1^T - u m_3^T) P = 0 \quad (m_2^T - v m_3^T) P = 0 \quad \rightarrow \quad \begin{bmatrix} P_i^T & \emptyset^T & -u_i P_i^T \\ \emptyset^T & P_i^T & -v_i P_i^T \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

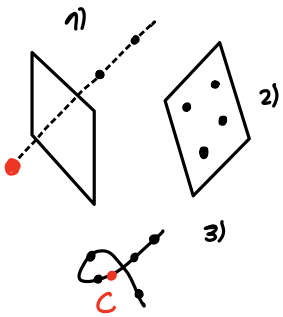
- We can stack N points : $\begin{bmatrix} P_1^T & 0 & -u_1 P_1^T \\ 0 & P_1^T & -v_1 P_1^T \\ P_2^T & 0 & -u_2 P_2^T \\ 0 & P_2^T & -v_2 P_2^T \\ & \vdots & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = Q \cdot M = 0$ $M \in \mathbb{R}^{12}$ (vectorized matrix)
  Known

- $M \in \mathbb{R}^{4\times 3}$: 12 unknowns
- $Q \in \mathbb{R}^{2n \times 12}$
- Each point provides 2 independent equations → need ≥ 6 points
- If $n \geq 6$ find least square solution:

$$M = \underset{\|M\|^2=1}{\arg\min} \|QM\|^2 = \underset{\|M\|^2=1}{\arg\min} M^T Q^T Q M \qquad \|M\|^2 = 1 \text{ prevents degenerate solution with } M = \emptyset$$

M^* is the eigenvector corresponding to the minimum eigenvalue

DEGENERATE CONFIGURATIONS



1. Points lying on a PLANE
2. Points along a single LINE PASSING THROUGH THE CENTER OF PROJECTION
3. Camera and points on a TWISTED CUBIC (smooth curve in 3D of degree 3)
4. Better use >20 points non coplanar

RECOVER K, R, T FROM M

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = \begin{bmatrix} \alpha r_{11} + u_0 r_{31} & \alpha r_{12} + u_0 r_{32} & \alpha r_{13} + u_0 r_{33} & \alpha t_1 + u_0 t_3 \\ & & & \vdots \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

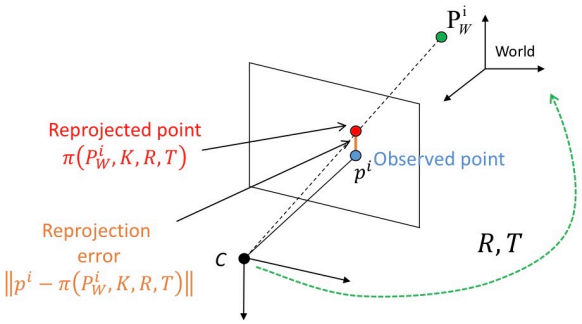
Not ensuring $R = R^T$

- We can apply the QR factorization of M: $M = K \cdot R \cdot T$
  upper orthogonal triangular

RECONSTRUCTION ERROR

- The REPROJECTION ERROR is the euclidean distance between the observed point and the projected point (using estimated K, R, T).

- Refine K, R, T and introduce lens distortion solving non-linear optimization using DLT solution as guess



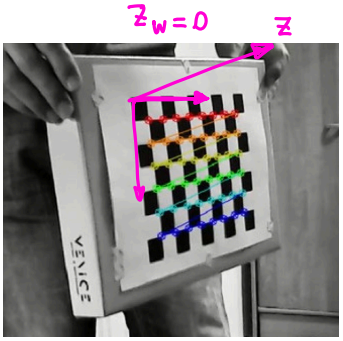
$$K, R, T, dist = \underset{K,R,T,dist}{\arg\min} \sum_{i=1}^{N} \|p^i - \pi(P_w^i, K, R, T)\|^2$$

N.B. Levenberg-Marquardt method is more robust than Gauss-Newton to local minima

ZHANG'S METHOD

- Tsai's method drawback: 3D points are not coplanar
- Todays calibration: multi view of a planar grid.

- In Zhang's method $Z_w = 0$



$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R|T] \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} h_1^T \\ h_2^T \\ h_3^T \end{bmatrix}}_{H} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

H: homography

→ Apply DLT for a single view

$$\begin{aligned} (h_1^T - u_i h_3^T) P_i &= 0 \\ (h_2^T - v_i h_3^T) P_i &= 0 \\ &\vdots \\ \forall\, i \in [1, n] & \end{aligned} \quad \Rightarrow \quad \underbrace{Q}_{\text{Known}} \cdot H = 0$$

- $Q \in \mathbb{R}^{2n \times 9}$, $H \in \mathbb{R}^9$
- each point provides 2 indep. equations
- At least 4 non collinear points required (more the better → use SVD as before)

RECOVER K, R, T

- multiple views needed (20-50 spanning the camera FoV)
- each view has the same K, but different R, T $\qquad H^j = K \cdot [R|t]^j$

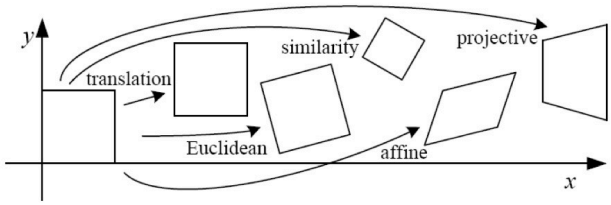
Algorithm

1. Estimate H_i $\forall$ i^{th} view using DLT algorithm
2. Determine K from a set of homographies
   - $\forall$ omography
     $H_i = K[r_1 | r_2 | t] := [w_1 | w_2 | t]$
   - From orthogonality
     $r_1^T r_2 = \underbrace{r_1^T K^T}_{w_1^T} \underbrace{K^{-T} K^{-1}}_{:=B} \underbrace{K r_2}_{w_2^T} = w_1^T B w_2 = 0$
     $r_1^T r_1 = r_2^T r_2 = 1 \quad = w_1^T B w_1 - w_2^T B w_2 = 0$
     } 2 equation $\forall$ homography
   - Stack 2n equations for n views and estimate B
3. Determine K from B using Cholesky
4. Determine R, T $\forall$ homography

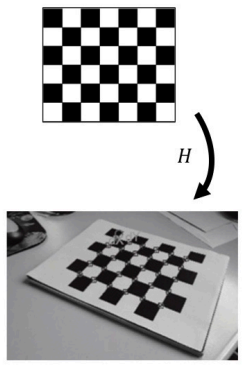
$K r_1 = H_1$ (first column) $\qquad KT = H_3$

$K r_2 = H_2$ $\qquad$ and $r_3 = r_1 \times r_2$

Types of 2D Transformations



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{2\times3}$	2	orientation + ···	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{2\times3}$	3	lengths + ···	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{2\times3}$	4	angles + ···	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times3}$	6	parallelism + ···	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times3}$	8	straight lines	



This matrix is called **Homography**

Projective Transformation (Homography)

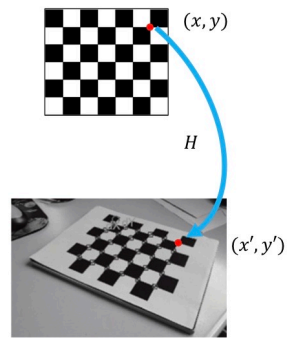
- A point (x, y) is transformed into (x', y') via:

$$x' = \frac{a_1 x + a_2 y + a_3}{a_7 x + a_8 y + 1}$$

$$y' = \frac{a_4 x + a_5 y + a_6}{a_7 x + a_8 y + 1}$$

- Homogeneous coordinates:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

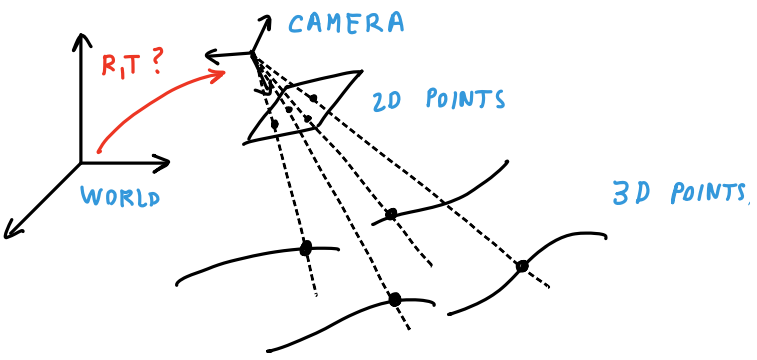


CAMERA LOCALIZATION (PERSPECTIVE FROM n POINTS = PnP)

PROBLEM: Determine 6 DoF pose of camera with respect to the world frame from a set of 3D-2D points correspondences

ASSUMPTIONS: Camera intrinsics are known

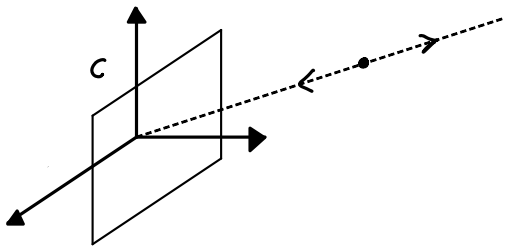
SOLUTIONS: DLT can be used but suboptimal. It exists an algebraic solution.



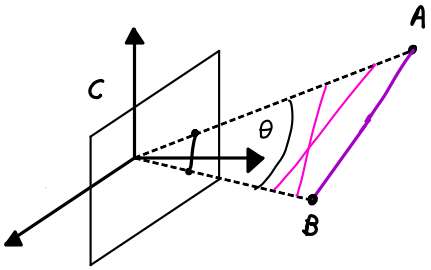
HOW MANY POINTS ARE ENOUGH?

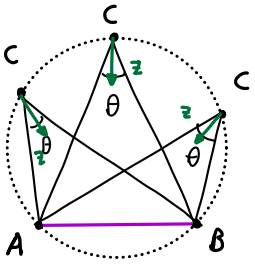
- 1 POINT → ∞ solutions
- 2 POINTS → ∞ solutions but bounded
- 3 POINTS → (if non collinear) up to 4 solutions
- 4 POINTS → unique solution

1 POINT



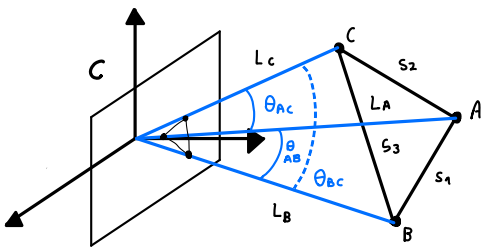
2 POINTS





bounded → camera must lie on that circle but ∞ solutions

3 POINTS



Unknown: L_A, L_B, L_C

Known: $s_1, s_2, s_3, \theta_{AB}, \theta_{BC}, \theta_{AC}$

From law of cosines

$$s_1^2 = L_B^2 + L_A^2 - 2L_A L_B \cos\theta_{AB}$$
$$s_2^2 = L_A^2 + L_C^2 - 2L_A L_C \cos\theta_{AC}$$
$$s_3^2 = L_B^2 + L_C^2 - 2L_B L_C \cos\theta_{BC}$$

FACTS

- n independent polynomials in n unknowns can have no more solutions than the product of their respective degree
- If every term is constant or of degree 2, for each positive solution there is one negative

P3P Problem: 3 eq. of degree 2 → $2 \cdot 2 \cdot 2 = 8$ solutions
terms constant or degree 2 → 4 solutions are positive
4 solutions are negative (non valid)
⇒ 4 solutions

- It can be shown that defining $x = L_B/L_A$ the system can be reduced to

$$G_0 + G_1 x + G_2 x^2 + G_3 x^3 + G_4 x^4 = 0$$

- A 4th point helps to disambiguate solutions. Their classification and recovering R and T with Gao's algorithm (OpenCV: solvePnP P3P)

MODERN SOLUTIONS TO PnP

- AP3P
- EPnP (efficient PnP): 10× more efficient and accurate than DLT (when camera is calibrated)

APPLICATIONS

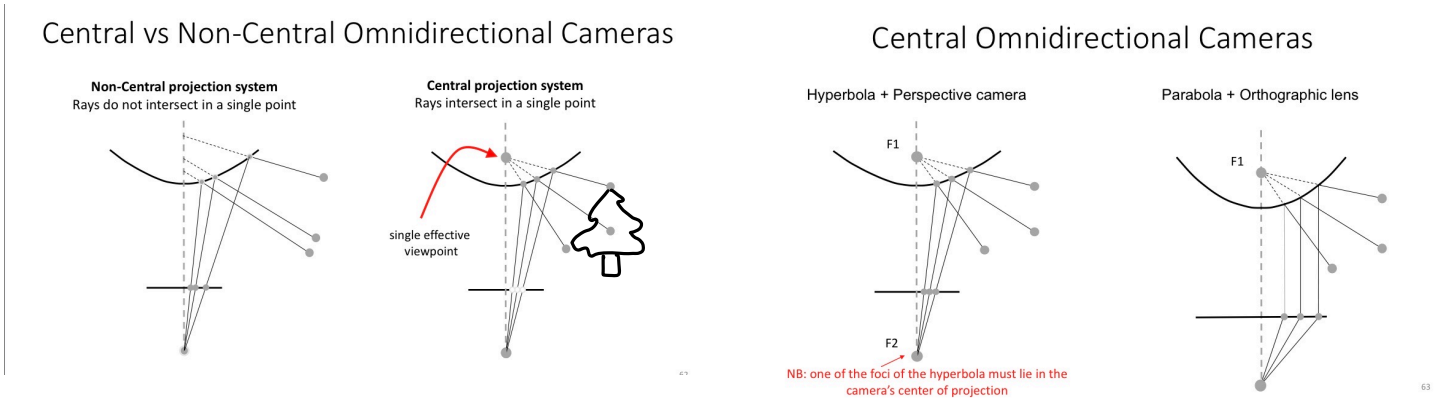
- LOCALIZATION: given a PCL get camera pose
- MONOCULAR VISUAL ODOMETRY: given initial point cloud find correspondences and new camera pose

ROBUSTNESS TO OUTLIERS

- All PnP problems are error prone if outliers in 3D-2D correspondences
- The RANSAC algorithm can be used to remove outliers
- PnP + Ransac in OpenCV: solvePnPRansac
- The output of both EPnP and DLT can be refined via non linear optimization

NON CONVENTIONAL CAMERA MODELS

FISHEYE : wide FOV (≥ 130°)
CATADIOPTRIC : 360° all around



CENTRAL CAMERAS are better

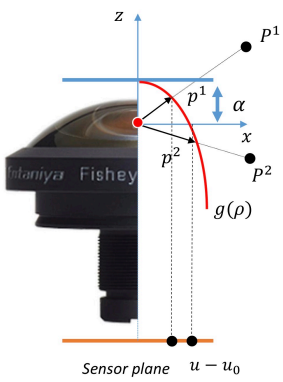
- We can apply standard algorithms valid for perspective geometry
- unwarp part of the image in a perspective one
- transform image points into normalized vectors on the unit sphere

UNIFIED OMNIDIRECTIONAL CAMERA MODEL (for fisheye and catadioptric cameras)

- define a image projection function with polynomials whose coefficients need to be estimated
- found coeff + intrinsics + extrinsics via DLT.

$$\frac{\lambda}{\alpha}\begin{bmatrix} u-u_0 \\ v-v_0 \\ g(\rho) \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \qquad g(\rho) = \alpha + \sum_{i=1}^{N} a_i \rho^i$$

$a_i, i>0 = 0 \longrightarrow$ pinhole camera



- OCamCalib: toolbox for calibrating wide angle camera

IMAGE FILTERING

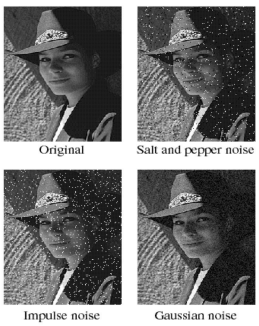
- FILTER : process that remove certain "frequency components"
- LOW-PASS FILTER : smooth image (keeps low-frequency)
- HIGH-PASS FILTER : retains contours (keeps high-frequency = sharp intensity discontinuities

NOISE REDUCTION

- SALT AND PEPPER NOISE : random black and white pixels
- IMPULSE NOISE : random white pixels
- GAUSSIAN NOISE : intensity variation drawn from Gaussian

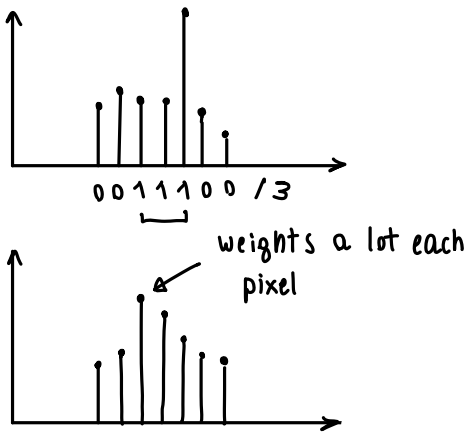
$$I(x,y) = I'(x,y) + \eta(x,y)$$
$$\eta(x,y) \sim N(0,\sigma)$$

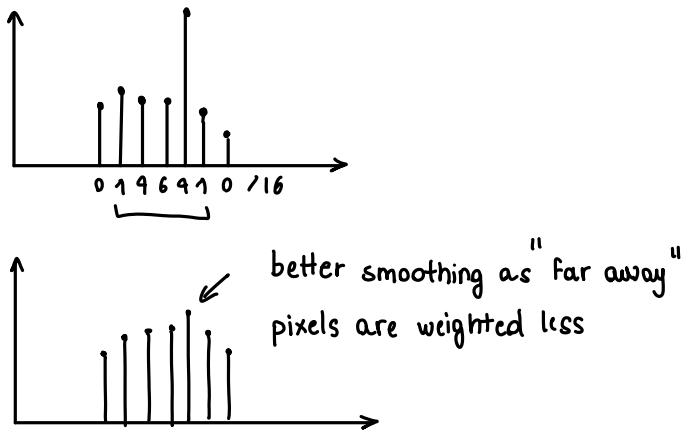
I' : ideal image



MOVING AVERAGE

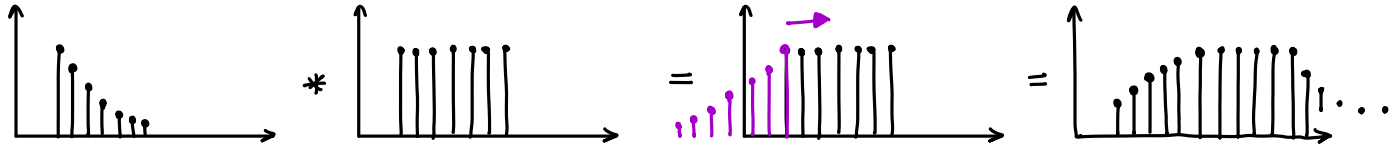
- One possible method to reduce Gaussian noise
- Assuptions :
  1) pixels are like neighbours
  2) noise independent from pixel to pixel



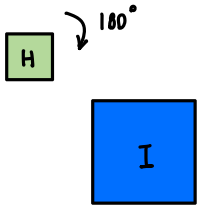


CONVOLUTION

- One of the sequences is flipped (right to left) before sliding over the other
- Notation : $a * b$
- Properties : linearity, associativity, commutativity

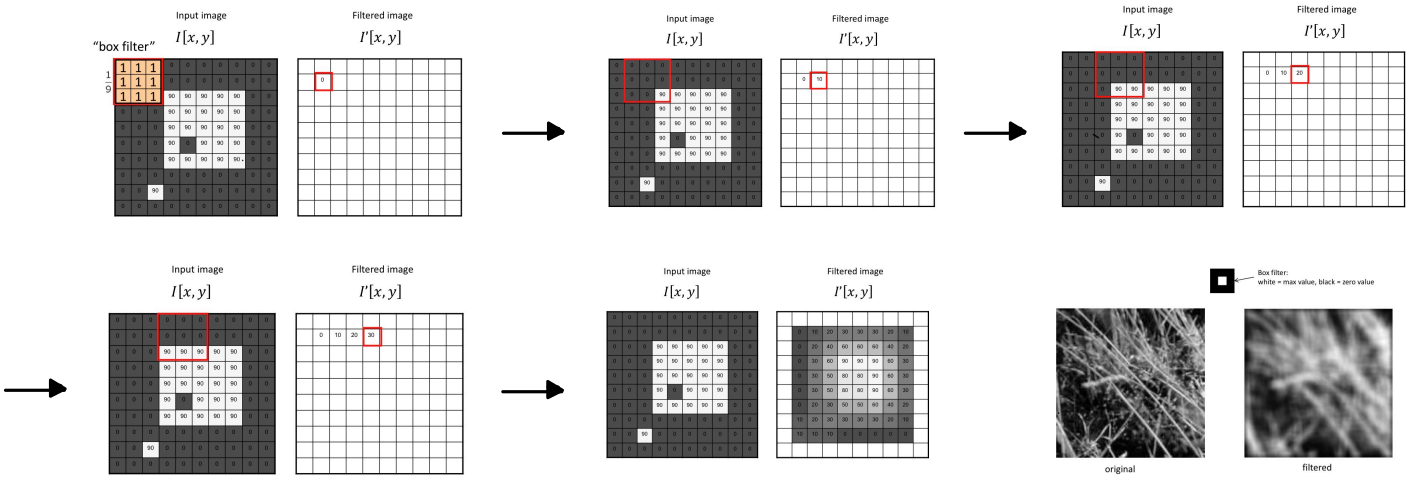


2D FILTERING VIA 2D CONVOLUTION



- Flip the filter in both dimensions (bottom to top, right to left = 180° turn)
- Then slide filter over the image and compute sum products
- Convolution replaces each pixel with a weighted sum of its neighbors
- The filter H is also called kernel or mask

- **CONVOLUTION** : $I' = I * H$ (linearity, associativity, commutativity) $\quad I'[x,y] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} I[x-u, y-v] H[u,v]$
- **CROSS-CORRELATION** : $I' = I \otimes H$ (linearity but not associativity, commutat.) $\quad I'[x,y] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} I[x+u, y+v] H[u,v]$
  - The same if H is symmetric

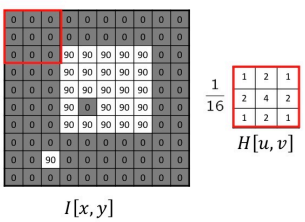


NOTE : the box filter causes a "web-like" effect because of "aliasing": high frequency components seen as low frequency

GAUSSIAN FILTER

This filter is an approximation of a Gaussian 2D function $H[u,v] = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$

- **Kernel size** : size of the filter → the larger, the better its discrete approximation
- **Variance** : controls the amount of smoothing
  standard deviation = σ [pixels]
  variance = σ^2 [pixels^2]



SEPARABLE FILTERS

- A convolution with a 2D filter of w×w pixel size requires w^2 multiply-add operations per pixel
- 2D convolution can be sped up if the filter is separable :
- A separable filter can be written as the product of 2 1-D filters : $H = v \cdot h^T$

$$I' = I * H = (I * h^T) * v \quad \rightarrow \text{do 2 1D convolutions only} = 2w \text{ multiply-add operations per pix.}$$

Examples:

- Box Filter $= \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} \cdot \frac{1}{3}[1\ 1\ 1]$
- Gaussian Filter $= \frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} = \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} \cdot \frac{1}{4}[1\ 2\ 1]$
- Sobel filter $= \begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix} = \begin{bmatrix}1\\2\\1\end{bmatrix} \cdot [-1\ 0\ 1]$

BOUNDARY ISSUES

What do we do when the filter goes through the edge? We pad with different methods:

- zero padding (black contour)
- wrap around
- copy edge
- reflect across edge

NON LINEAR FILTERS

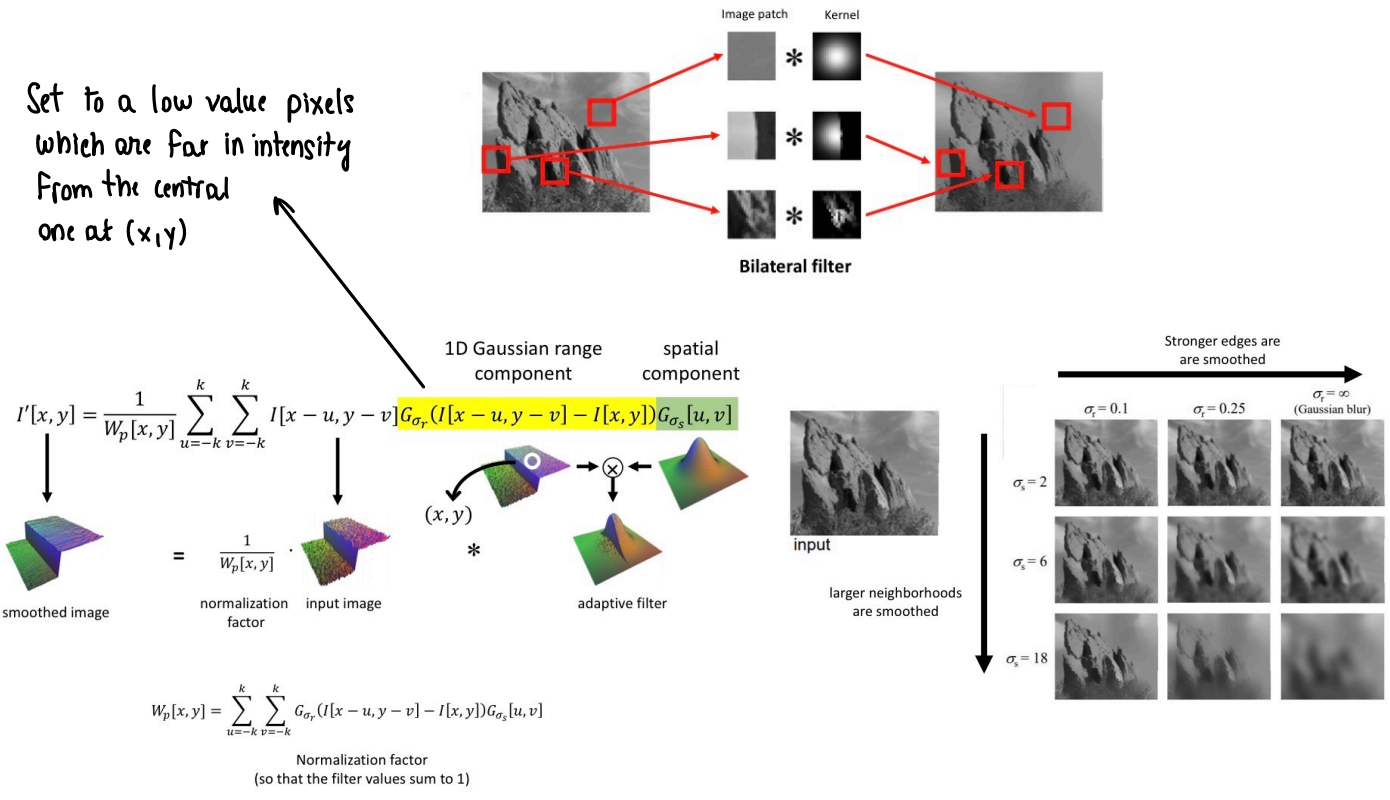
- Linear filters cannot remove effects like salt-and-pepper noise.
- Non-linear filters like the median filter can.

MEDIAN FILTER

- It's a non linear filter : sort the input patch and extract the median value
- Removes spikes : good for salt-and-pepper and impulse noise.
- Preserves strong edges and does not smooth (contrary to Gaussian filter)

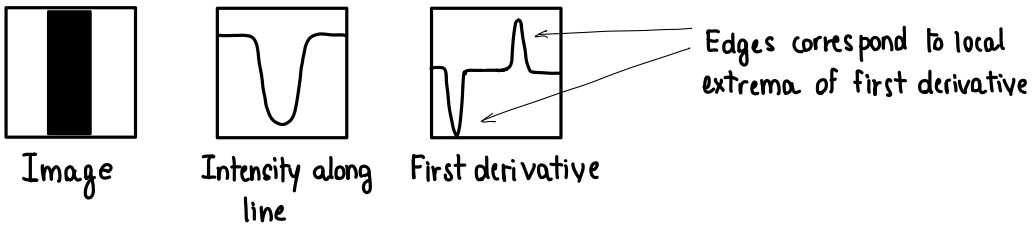
BILATERAL FILTER

- Solves the last problem by adapting the kernel locally to the intensity profile, so they are patch-content dependent
- They smooth only pixels with the same brightness as the center pixel and ignore influence of pixels with different brightness across the discontinuity



EDGE DETECTION

- An edge is a place of fast change in the image intensity function



$$\frac{\partial I(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{I(x+\varepsilon,y) - I(x,y)}{\varepsilon} \approx \frac{I(x+1,y) - I(x,y)}{1} \quad \text{adjacent derivative}$$

$$\approx \frac{I(x+1,y) - I(x-1,y)}{2} \quad \text{central derivative}$$

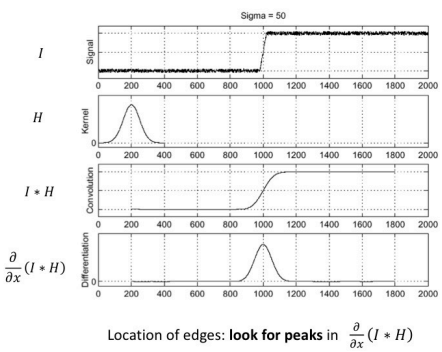
Central derivative filters:

- Prewitt filter $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$, $G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
- Sobel filter $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$, $G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

IMAGE GRADIENT

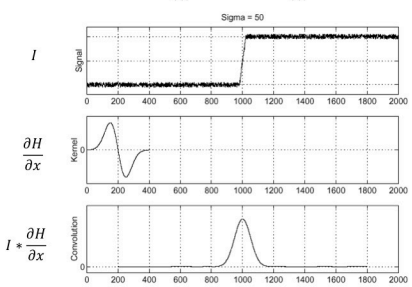
- The image gradient $\nabla I = [\partial I/\partial x, \partial I/\partial y]$
- The gradient direction is $\theta = \text{atan2}(\partial I/\partial y, \partial I/\partial x)$
- The gradient points in the direction of steepest ascent
- The edge strength is given by the gradient magnitude $\|\nabla I\|$

- noise has the effect of making gradient noisy → edge difficult to find
- solution 1: smooth first and find peak of gradient



Location of edges: **look for peaks** in $\frac{\partial}{\partial x}(I * H)$

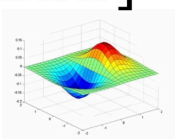
- Differentiation property of convolution: $\frac{\partial}{\partial x}(I * H) = I * \frac{\partial H}{\partial x}$



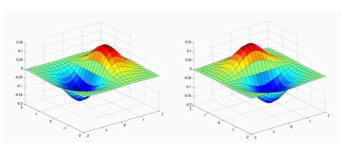
Derivative of Gaussian filter G along x

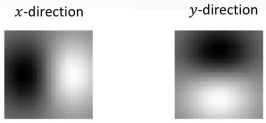
$$(I * G) * H = I * (G * H)$$

$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} * \begin{bmatrix} -1 & 1 \end{bmatrix}$$



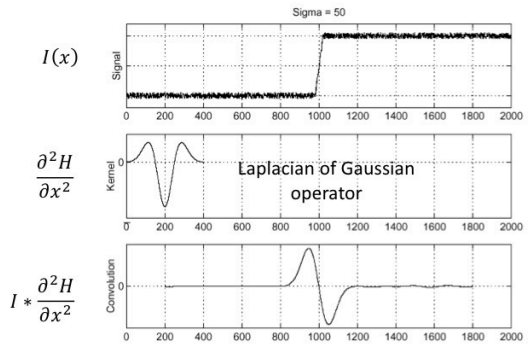
Derivative of Gaussian Filters



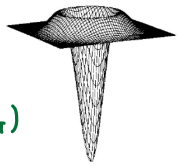


- solution 2: find zero-crossing of second order derivative (where there is a sign change = derivative change) of the intensity

$$\frac{\partial^2}{\partial x^2}(I * H) = I * \frac{\partial^2 H}{\partial x^2}$$



Location of edges: look for **Zero-crossings** of $I * \frac{\partial^2 H}{\partial x^2}$



LAPLACIAN OF GAUSSIAN (LoG)

- It is a circularly simmetric filter defined as:

$$\nabla^2 G_\sigma = \frac{\partial^2 G_\sigma}{\partial x^2} + \frac{\partial^2 G_\sigma}{\partial y^2} \qquad \nabla^2 \triangleq \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

- two commonly used approximations

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

RECAP FILTERS

Smoothing filters
- has positive values
- sums to 1 → preserve brightness of constant regions
- removes "high frequency" components

Derivative filters
- has opposite signs used to get high response in regions of high contrast
- sums to 0 → no response in constant regions
- highlights "high frequency" components

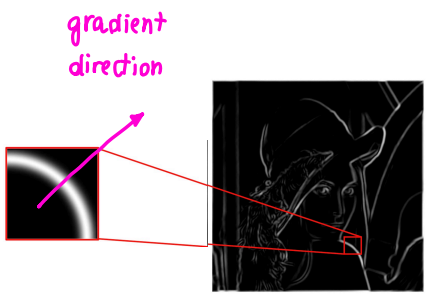
THE CANNY EDGE DETECTION ALGORITHM

1. Take a grayscale image. If RGB, take mean of its channels
2. Convolve the image I with x and y derivative of Gaussians filter and compute the edge strength $\|\nabla I\|$
3. Thresholding: set to 0 all pixels where $\|\nabla I\| <$ threshold
4. Thinning: look for local maxima in the edge strength in the direction of the gradient
   - take the directional derivative of the edge strength in the direction of the gradient and then looking for zero crossing (adjacent pixels where sign change = edge raising or falling)
   - Equivalent to $\nabla^2 G_\sigma * I$

1

2

3

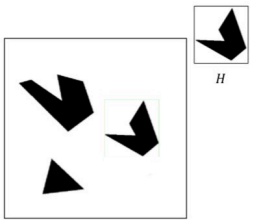


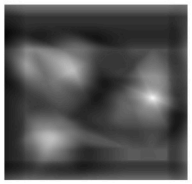
4

FILTERS FOR FEATURE DETECTION

Filters can be used to extract features
- keep what is important
- reduce later processing
- reduce memory requirements and compute

Examples are
- edge detection
- template matching
- keypoint detection





Correlation map I':

$$I'[x,y] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} I[x+u, y+v] H[u,v]$$

TEMPLATE MATCHING

- Find locations in an image I that are similar to a template H
- We can use cross-correlation (180° rotated filter)
- Will work only if scale, orientation, illumination and in general the appearance of the template (including background) are very similar.

Correlation can be seen as scalar product between two vectorized image patches H and F: $\langle H,F \rangle = \|H\| \|F\| \cos\theta$

SIMILARITY MEASURES

Normalized Cross Correlation (NCC): $NCC = \cos\theta = \frac{\langle H,F \rangle}{\|H\| \|F\|} = \frac{\sum_u \sum_v H(u,v) F(u,v)}{\sqrt{\sum_u \sum_v H(u,v)^2} \sqrt{\sum_u \sum_v F(u,v)^2}}$

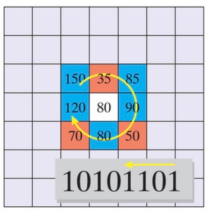
Sum of Squared Differences (SSD): $SSD = \sum_u \sum_v (H(u,v) - F(u,v))^2$

Sum of Absolute Differences (SAD): $SAD = \sum_u \sum_v |H(u,v) - F(u,v)|$

- To account for the difference in the average intensity of two images (typically caused by additive illumination changes) we subtract the mean value of each patch: $\mu_H = \frac{1}{n} \sum_u \sum_v H(u,v)$; $\mu_F = \frac{1}{n} \sum_u \sum_v F(u,v)$
  We than replace H and F in the previous formula with $H' = H - \mu_H$ and $F' = F - \mu_F$ obtaining the zero zero-mean version of the metrics: ZNCC, ZSSD and ZSAD.
- Note that ZNCC is invariant to affine illumination changes: $I' = \alpha I + \beta$, the others not.

CENSUS TRANSFORM

- Maps an image patch to a bit string
  - If a pixel is greater than the center pixel its corresponding bit is set to 1, else 0
  - For a $w \times w$ patch the string will be $w^2 - 1$ long
- The two strings are compared using the Hammin distance (number of different bits = Σ XOR)
- No square roots or divisions → efficient (especially on FPGA)
- Intensities relative to the center pixel → invariant to monotonic intensity changes.



KEYPOINTS EXTRACTION AND MATCHING

Why? They are the feature we extract and match between consecutive frames to estimate motion. But Keypoints are also used for:

- Panorama stitching
- Object recognition
- 3D reconstruction
- Place recognition
- Indexing and database retrieval

} **Feature Matching Problem**: find similar keypoints between two images of the same scene taken under **different conditions**.

PROBLEM 1
Can we **redetect** features?

Repetability: property of a "detector" to re-detect the same feature in different images of the same scene.

PROBLEM 2
Can we **match** te same feature?

Distintiveness: a descriptor uniquely identifies a feature from other features without ambiguity. Must be **robust** to **geometric and photometric changes**

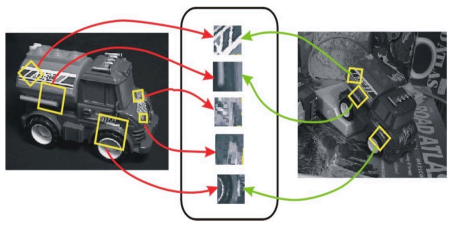
- Geometric changes: rotation, scale, view point (perspective change)
- Photometric changes: affine intensity transformations:

$$I'(x,y) = \alpha I(x,y) + \beta$$

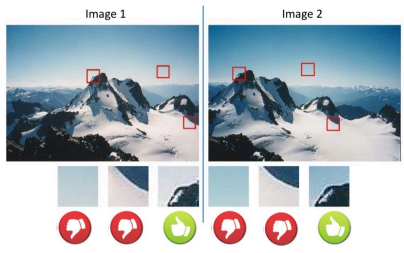
LOCAL INVARIANT FEATURES

Try to address these problems and be repeatable, distinctive and robust:

1. Detect repeatable and distinctive **interest points**;
2. Extract **invariant** (robust) **descriptors**.



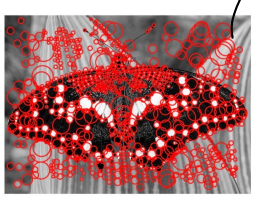
REPEATABLE AND DISTINTIVE FEATURE?



POINT FEATURES: CORNERS VS BLOB DETECTORS

Corner detector: intersection of two or more edges
(+) **high localization accuracy**: good for VO
(-) **less distinctive than blobs**
Ex.: Harris, Shi-Tomasi, SUSAN, FAST

Blob detector: any other image pattern that **is not a corner** and differs significantly from its neighbors
(+) blobs have **less localization accuracy** than corners
(-) blobs are **more distinctive**: better for Place Recognition
Ex.: MSER, LOG, DOG (SIFT), SURF, CenSurE, etc.

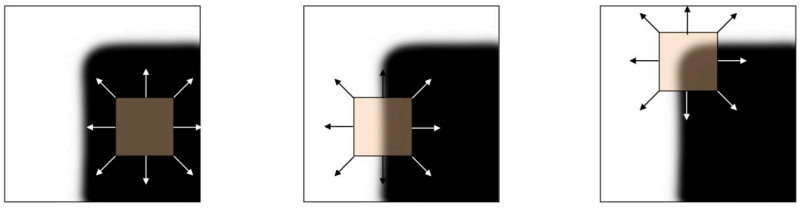


→ Connected region of pixels with similar color, a circle, etc...

CORNER DETECTION

- In the region around a corner, image gradients has two or more dominant directions.
- How?
  - We look at a region of pixels through a small window.
  - Shifting the window in any direction should cause large intensity changes (i.e. in SSD*)

*Sum of Squared Distances



"flat" region: no intensity change (i.e., SSD ≈ 0 in all directions)

"edge": no change along the edge direction (i.e., SSD ≈ 0 along edge but ≫ 0 in other directions)

"corner": significant change in all directions (i.e., SSD ≫ 0 in all directions)

THE MORAVEC CORNER DETECTOR (1980)

- Consider the reference patch centered at (x,y) and the shifted window centered at $(x+\Delta x, y+\Delta y)$. The patch has size Ω. The Sum of Squared Distances is

$$SSD(\Delta x,\Delta y) = \sum_{x,y\in\Omega} (I(x,y) - I(x+\Delta x, y+\Delta y))^2$$

- Repeat this for the 4 directions (right, down, down right, down left = horizontal, vertical and two diagonals).
- The window's interest measure is the minimum of these 4.
- Features are chosen where interest measure has local maxima.
- Problem: 4 computations every time ...

THE HARRIS CORNER DETECTOR (1988)

- Implements Moravec without shifting but using derivative filters.
- The shifted image can be approximated with a 1st order Taylor expansion

$$I(x+\Delta x, y+\Delta y) \approx I(x,y) + I_x(x,y)\Delta x + I_y(x,y)\Delta y \Rightarrow SSD = \sum_{x,y\in\Omega} (I(x,y) - I(x+\Delta x, y+\Delta y))^2$$

$$= \sum_{x,y\in\Omega} (I_x(x,y)\Delta x + I_y(x,y)\Delta y)^2$$

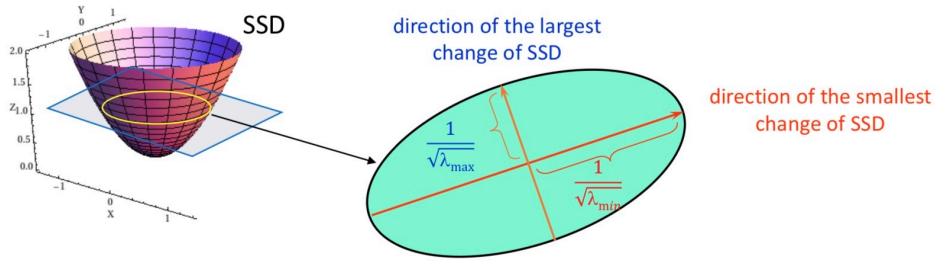
$$= \sum_{x,y\in\Omega} \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}$$

$$SSD(\Delta x,\Delta y) \approx \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} \begin{bmatrix} \sum_\Omega I_x^2 & \sum_\Omega I_xI_y \\ \sum_\Omega I_xI_y & \sum_\Omega I_y^2 \end{bmatrix} \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}$$

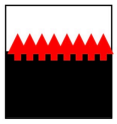
Quadratic form

M (2nd moment matrix): I_xI_y are not matrix products but pixel-wise products.

- Since M is symmetric, it can always be decomposed into $M = R^{-1}\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}R$
- We can visualize $[\Delta x\ \Delta y]M\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = \text{const}$ as an ellipse with axis length λ_1, λ_2 and orientation R.
- The two eigenvectors identify the directions of largest and smallest changes of SSD.



Example

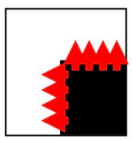


Edge

$$M = \begin{bmatrix}\sum I_x^2 & \sum I_x I_y\\ \sum I_x I_y & \sum I_y^2\end{bmatrix} = \begin{bmatrix}0 & 0\\ 0 & \lambda_2\end{bmatrix}, \lambda_1 \approx 0$$

Flat region

$$M = \begin{bmatrix}\sum I_x^2 & \sum I_x I_y\\ \sum I_x I_y & \sum I_y^2\end{bmatrix} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}, \lambda_1 \approx 0 \wedge \lambda_2 \approx 0$$



Corner

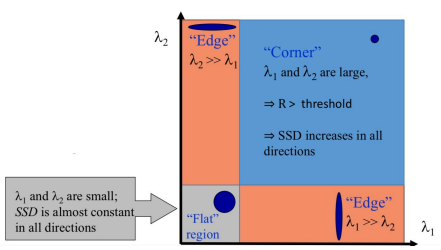
$$M = \begin{bmatrix}\sum I_x^2 & \sum I_x I_y\\ \sum I_x I_y & \sum I_y^2\end{bmatrix} = \begin{bmatrix}\cos\frac{\pi}{4} & -\sin\frac{\pi}{4}\\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4}\end{bmatrix}\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}\begin{bmatrix}\cos\frac{\pi}{4} & \sin\frac{\pi}{4}\\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4}\end{bmatrix}, \lambda_1 \gg 0 \wedge \lambda_2 \gg 0$$

REVIEW: COMPUTE λ_1, λ_2, R FROM M

- $Ax = \lambda x$ for A square
- Non trivial eigenvalues ($x = \vec{0}$) solving $\det(A - \lambda I) = 0$
- A = M: $\det\begin{bmatrix}m_{11}-\lambda & m_{12}\\ m_{21} & m_{22}-\lambda\end{bmatrix} = 0 \rightarrow \lambda_{1,2} = \frac{1}{2}\left[(m_{11}+m_{22}) \pm \sqrt{4m_{12}m_{21} + (m_{11}-m_{22})^2}\right]$
- Once λ is known, you find the two eigenvectors (two columns of R) solving $\begin{bmatrix}m_{11}-\lambda & m_{12}\\ m_{21} & m_{22}-\lambda\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix} = 0$

INTERPRETING EIGENVALUES

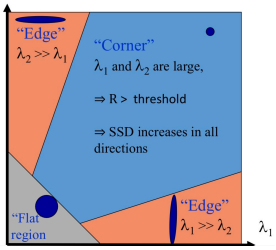
- A corner can be identified using a threshold: $R = \min(\lambda_1, \lambda_2) > \text{threshold}$
- R is called cornerness function
- The corner detector using this criterion is called "Shi-Thomasi" detector (1994).



- Computation of λ_1 and λ_2 is expensive → Harris and Stephens suggested using a different cornerness function:

$$R = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2 = \det(M) - k\,\mathrm{trace}^2(M) \quad \text{with } k \text{ "magic number"} \in [0.04, 0.15]$$

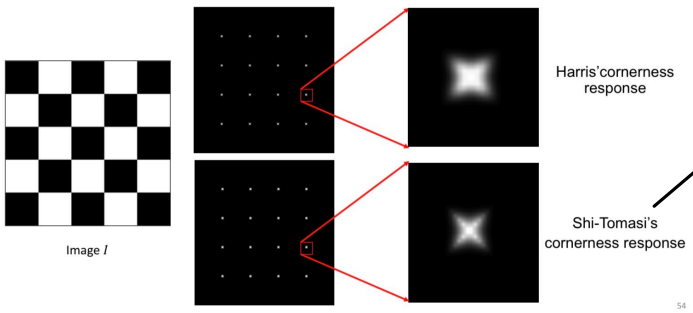
- The corner detector using this criterion is called "Harris" detector (1988).



CORNER DETECTOR ALGORITHM (FULL)

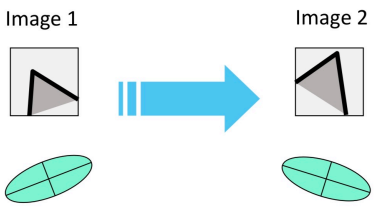
1. Compute derivatives in x and y (I_x, I_y) e.g with Sobel filter.
2. Compute I_x^2, I_y^2 and I_xI_y
3. Convolve I_x^2, I_y^2 and I_xI_y with a box filter to get ΣI_x^2, ΣI_y^2 and ΣI_xI_y which are the entries of M (optionally use a Gaussian filter to avoid aliasing and give more weight to the central pixel).
4. Compute cornerness R
5. Threshold (R > threshold)
6. Take the points of local maxima of R (Non-Maxima Suppression).

HARRIS VS SHI-THOMASI



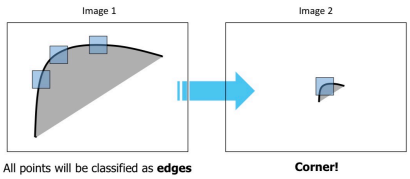
→ Precise maximum: peaked response and precise localization

- Harris detector is invariant to rotation



- Harris detector is NOT invariant to scaling

$$\text{Repeatability} = \frac{\#\text{ correspondences detected}}{\#\text{ correspondences present}}$$ drop with scaling



In this case it is important to tune correctly the Kernel size

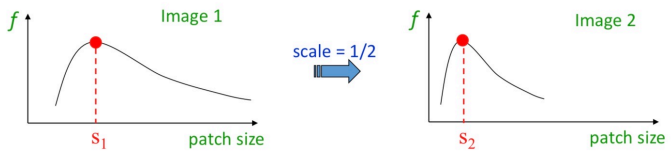
- Harris detector is invariant to affine illumination changes → $I' = \alpha I + \beta$ → $I_x' = \alpha I_x'$, $I_y' = \alpha I_y'$
  α can affect the threshold
- Harris detector is invariant to monotonic illumination changes → thanks to local non-maxima suppression
- Harris detector is invariant to "small" viewpoint changes

SCALE CHANGES

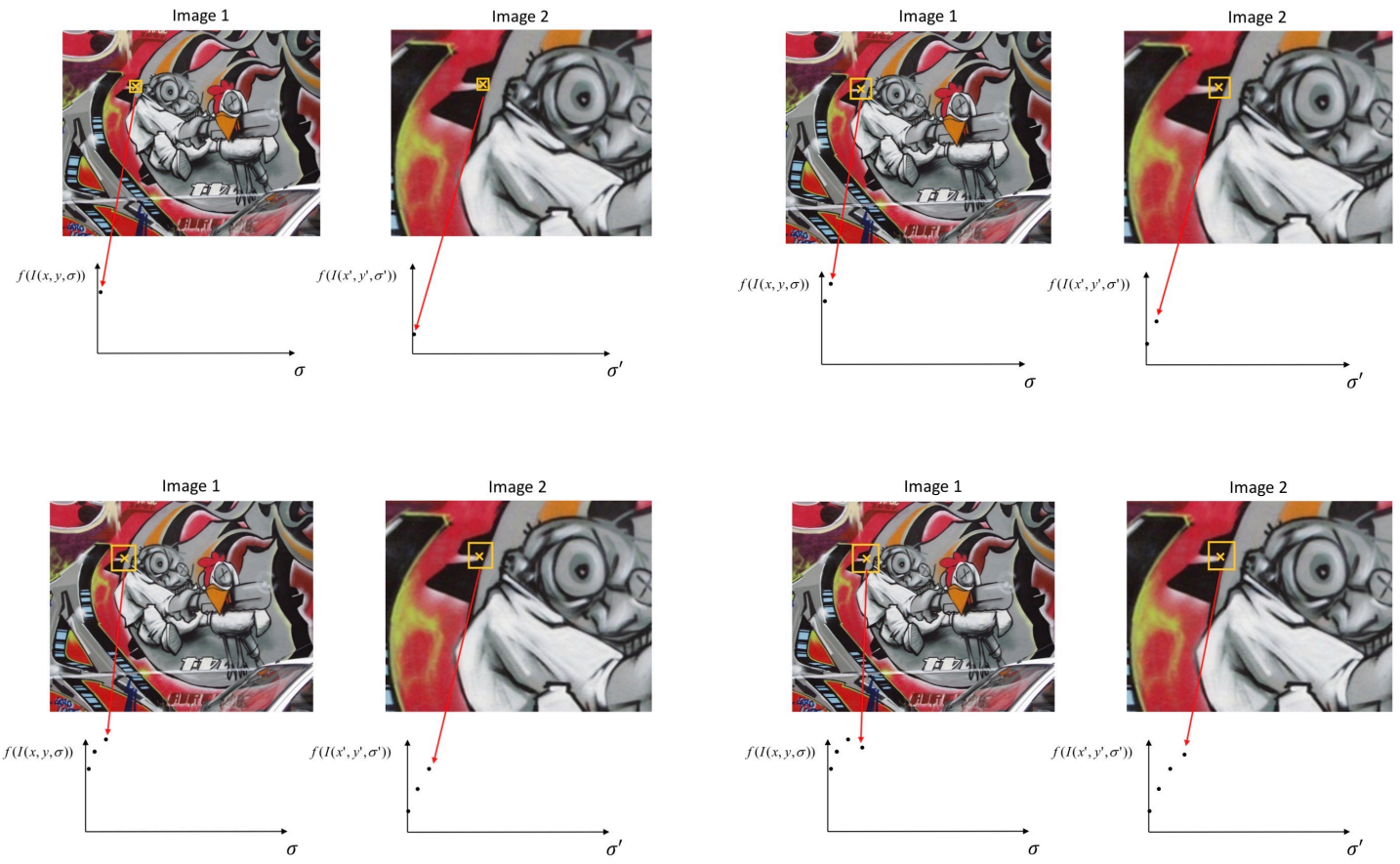
- Question? How can we scale image patches if they have different scale?
- Solution 1: Rescale the patch
  - Scale search is time consuming (needs to be done individually for all patches in one image)
  - Complexity is $(NS)^2$ assuming N features per image and S rescalings per feature
- Solution 2: automatic scale selection: automatically assign each feature its "own" scale

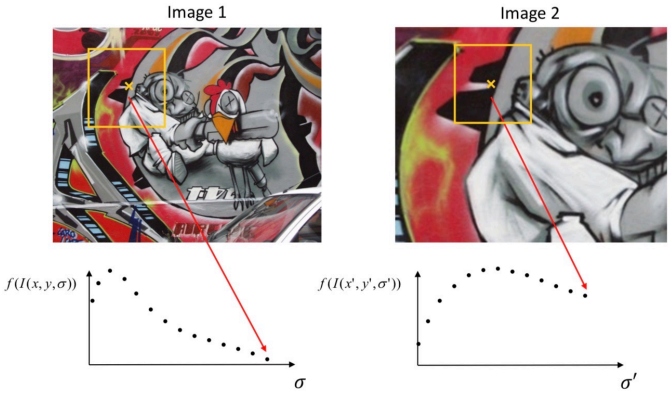
AUTOMATIC SCALE SELECTION

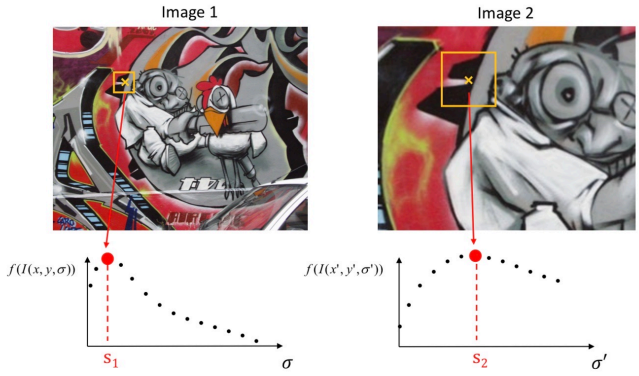
- Design a function on the image patch which is scale invariant (same value for corresponding patches even if they are at different scale)
- Find local extrema of this function
- The patch size at which the local extrema is reached should be invariant to image rescaling
- This scale invariant patch size is found in each image independently.

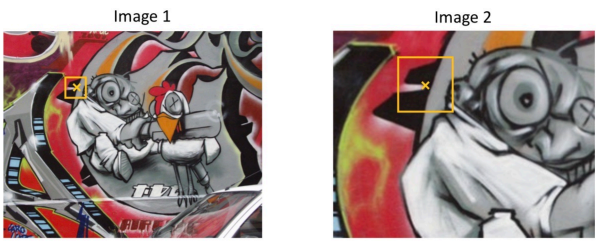


Example

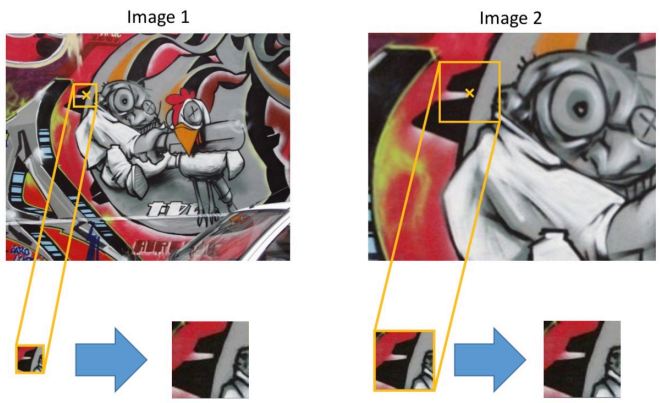








When the right scale is found, the patches must be **normalized to a canonical size** so that they can be compared by SSD



- A "good function" for scale detection should have a single & sharp peak

- The ideal function for determining the scale is one that highlights sharp discontinuities.
- The solution is to convolve image with a Kernel that highlights edges.
- It has been shown that the Laplacian of Gaussian is optimal under certain assumptions:

$$LoG(x,y;\delta) = \nabla^2 G_\delta(x,y) = \frac{\partial^2 G_\delta(x,y)}{\partial x^2} + \frac{\partial^2 G_\delta(x,y)}{\partial y^2}$$

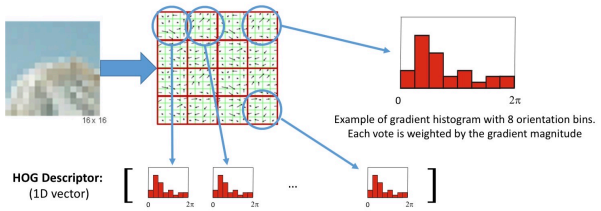
Works as a "blob" detector.

$$f = \text{Kernel} * \text{image}$$

- The correct scale is found as local extrema across consecutive smoothed patches (increase δ).

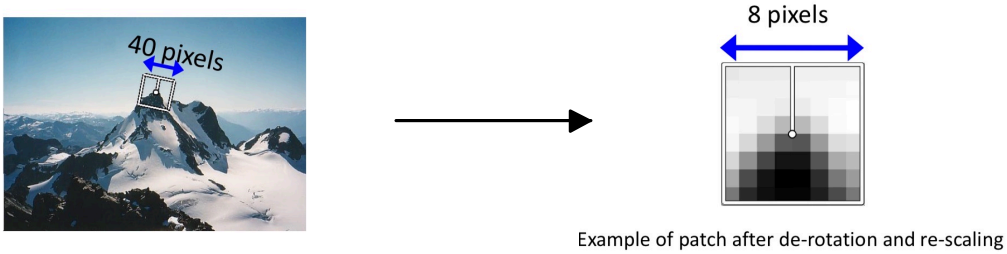
PATCH DESCRIPTORS

- "Naive" patch descriptor: patch of intensity (integer) values
- Census descriptor: see before (binary string)
- HOG (Histogram of Oriented Gradients) descriptor:
  - the image is divided into a grid of cells
  - for each cell a histogram of gradient directions is computed
  - the HOG descriptor is the concatenation of these histograms
  - differently from the previous two, it has float values



How to achieve rotation and scale invariance?

- Feature descriptors should be invariant to geometric and photometric changes
- De-rotation
  1. Determine the patch orientation (e.g eigenvectors of M from Harris detector or dominant gradient direction (see next)
  2. De-rotate patch through "patch warping" (see next). This puts the patch in a canonical orientation.
- Re-scaling
  1. Detect scale using the LoG operator
  2. Rescale the patch to a canonical size (e.g. 8×8 pixels).



Example of patch after de-rotation and re-scaling

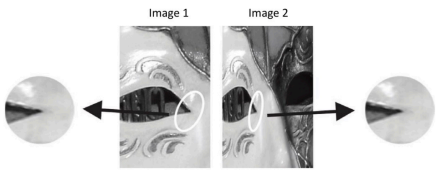
How to determine patch orientation?

1. Multiply the patch with a Gaussian kernel to make the shape circular
2. Compute gradients vectors at each pixel
3. Build a histogram of gradient orientations weighted by the gradient magnitude. This histogram is the HOG descriptor.
4. Extract all local maxima in the histogram: each local maximum above a threshold is a candidate dominant direction
5. Construct a different keypoint descriptor ∀ dominant direction.

How to achieve invariance to small view-point changes?

Affine warping provides invariance to small view-point changes

- The second moment matrix M of Harris can be used to identify the two directions of fastest and slowest SSD
- Out of these two directions an elliptic patch is extracted at the scale computed with the LoG operator.
- The region inside the ellipse is normalized to a canonical circular patch

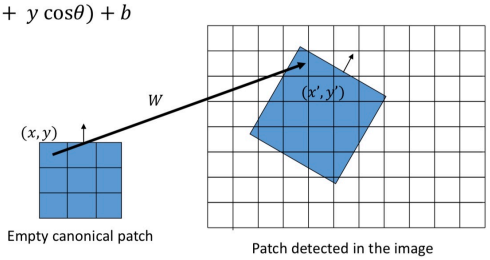


How to warp a patch?

- Start with an "empty" canonical patch (all pixels = 0)
- ∀ pixel (x,y) apply the warping function w(x,y) to compute the corresponding point in the source image. It will be in floating point and will fall between the image pixels.
- Interpolate the intensity values of the 4 closest pixels: nearest neighbour, bilinear, bicubic interpolation

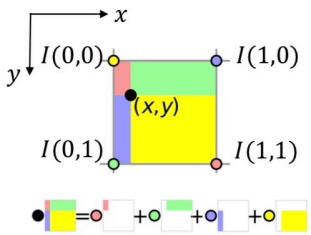
- Warping function W (counterclockwise rotation plus rescaling and translation):

$$x' = s(x\cos\theta - y\sin\theta) + a$$
$$y' = s(x\sin\theta + y\cos\theta) + b$$



BILINEAR INTERPOLATION

- Extension of linear interpolation in 2D
- Linear in each dimension but not a linear operator

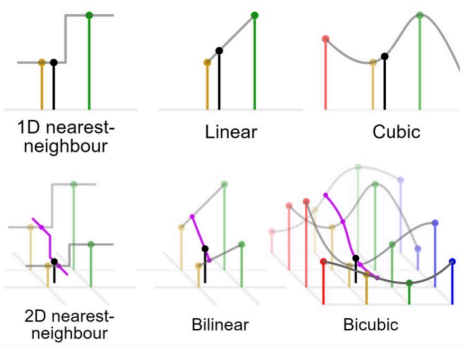


$$I(x,y) = I(0,0)(1-x)(1-y) + I(0,1)(1-x)(y) + I(1,0)(x)(1-y) + I(1,1)(x)(y)$$

Area of the opposite square

If $x = 0 \rightarrow I(0,y) = I(0,0)(1-y) + I(0,1)y$
If $y = 0 \rightarrow I(x,y) = I(0,0)(1-x) + I(1,0)x$

} reduces to 1D linear (Same for $x = 1$, $y = 1$)



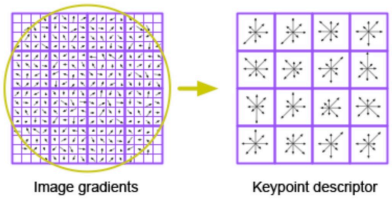
DISADVANTAGE OF PATCH DESCRIPTORS

- If the warp is not estimated accurately, very small errors in rotation, scale and view-point will affect matching score significantly.
- Computationally expensive: need to unwarp every patch.

SCALE INVARIANT FEATURE TRANSFORM (SIFT, 2004)

Descriptor computation:

- Multiply the patch by a Gaussian filter, compute dominant direction and de-rotate patch
- Consider a 16 × 16 pixel patch
- Compute HOG descriptor
  - Divide patch into 4×4 cells
  - Use 8 bins histograms
  - Concatenate all histograms into a single 1D vector
  - Resulting SIFT descriptor: 4×4×8 = 128 values



Why 4×4 and 8 bins? Later the answer.

DESCRIPTOR NORMALIZATION

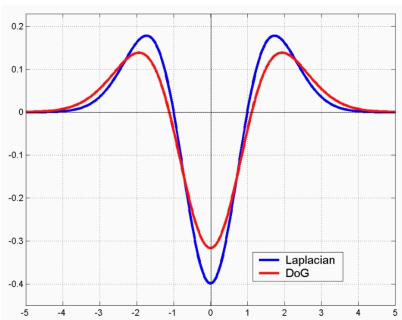
- The descriptor v is then normalized such that is ℓ_2 norm is 1: $\bar{v} = v / \|v\|$
- This guarantees that the descriptor is invariant to linear illumination changes.
- The descriptor is already invariant to additive illumination because it is based on gradients.

↳ It is invariant to affine illumination changes.

SIFT FACTS

- Can handle severe viewpoint changes (up to 50 deg out-of-plane rotations).
- Can handle even severe non affine changes in illumination (low to bright scenes).
- Computationally expensive: 10 fps on an i7 processor.
- SIFT uses the Difference of Gaussians kernel instead of Laplacian of Gaussians (LoG) because computationally cheaper

$$LoG(x,y) \approx DoG(x,y) = G_{k\sigma}(x,y) - G_{\sigma}(x,y)$$

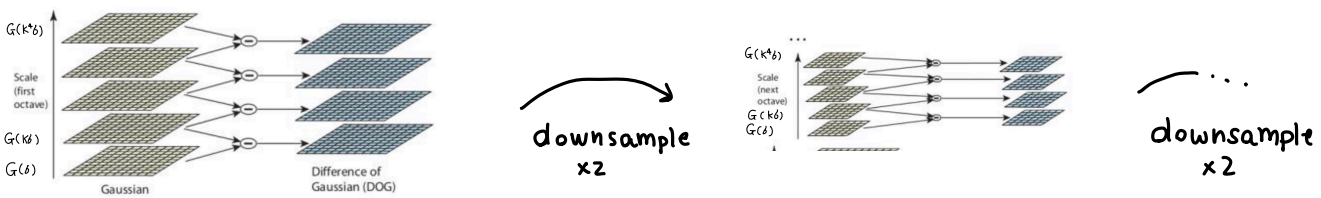


IMPLEMENTATION

1. Build a Space Scale Pyramid

The initial image is incrementally convolved with Gaussians $G(k^i\sigma)$ to produce blurred images separated by a constant factor k in scale space:

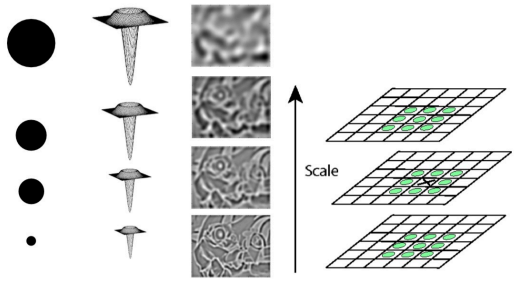
- The initial Gaussian $G(\sigma)$ has $\sigma = 1.6$.
- k is chosen $k = 2^{1/s}$ where s is the number of intervals into which each octave is divided.
- For efficiency, if $k^i = 2$, the image is downsampled by 2 and the procedure repeated 5 times.



2. Scale Space extrema detection

Detect local maxima and minima in scale spaces

- Each pixel is compared to 26 neighbors (in green below): its 8 neighbors in the current image + 9 neighbors in the adjacent upper and lower scale.
- If the pixel is a global maximum or minimum with respect to its 26 neighbors, then it is selected as SIFT features



SIFT RECAP

- A scale invariant feature transform
- An approach to detect and describe region of interest in an image.
- Invariant to 2D rotation and "reasonably" invariant to rescaling, viewpoint changes (up to 50°) and illumination
- Runs real-time but expensive (expensive steps are scale detection and descriptor extraction).
- SIFT outputs location of the patch, s
  - the full histogram info: $4 \times 4 \times 8 = 128$-element 1D vector,
  - location of the patch
  - scale (high scale = high blur = big blob)
  - orientation: angle of the patch.

FEATURE MATCHING

1. Define distance function ((Z)SSD, (Z)SAD, ...) or Hamming distance for binary descriptors
2. Brute-force matching
   - Compare each feature in I_1 with each feature in I_2 (N^2 comparisons)
   - Take the closest descriptor ↦ can return good score for false matches
   - Better approach: compute ratio of distances to 1st to 2nd closest descriptor

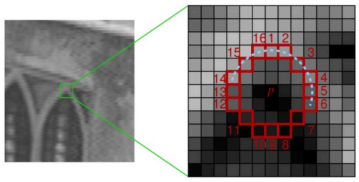
$$\text{accept if } \frac{d_1}{d_2} < \text{threshold } (\sim 0.8)$$

Why it works? If the closest is correct it is really correct while the second match is off. Instead when $d_1 \approx d_2$ then this is a noisy and ambiguous descriptor to match.

OTHER DESCRIPTORS

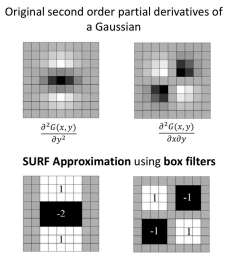
FEATURES FROM ACCELERATED SEGMENT TEST (FAST)

- Analyses intensities of 16 pixels around pixel p
- p is a FAST corner if a set of N contiguous pixels on the ring are brighter than the pixel intensity I(p) + threshold or all darker than I(p) − threshold (N : 12)
- A simple classifier is used to check the quality of the corners and reject the weak ones
- Fastest corner detector ever made : 100 milion pixels/s
- Very sensitive to noise (why Harris is still common) → in fact thought as pre-processing for Harris.



SPEEDED UP ROBUST FEATURES (SURF) BLOB DETECTOR

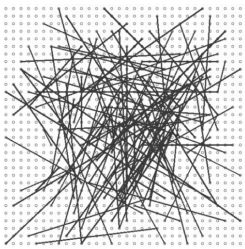
- Similar to SIFT but much faster
- Approximate DoG using box filters
- Results comparable with SIFT but
  - Faster computation
  - Generally shorter descriptors



BINARY ROBUST INDEPENDENT ELEMENTARY FEATURES (BRIEF) *

*can be applied to corner and blobs

- Goal: high speed computation and matching
- Binary description formation
  - smooth image
  - ∀ keypoint (e.g FAST)
    - sample 128 intensity pairs (p_1^i, p_2^i) $i \in [1, \dots, 128]$ within a squared patch around the keypoint
    - create an empty 128-element descriptor
    - ∀ i^{th} pair:
      - if $I_{p_1^i} < I_{p_2^i}$, set i^{th} bit of descriptor to 1, else 0.
- Pattern is generated randomly (or learned) and reused for all patches
- PRO: binary descriptor ↦ fast Hamming distance matching
- CON: not scale/rotation invariant.

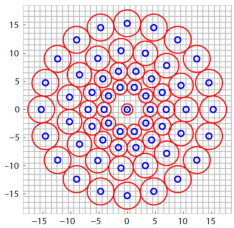

Pattern for intensity pair samples – generated randomly

ORIENTED FAST AND ROTATED BRIEF (ORB) *

- Keypoint descriptor originally based on FAST.
- Binary descriptor based on BRIEF but adds a orientation component to make it rotation invariant

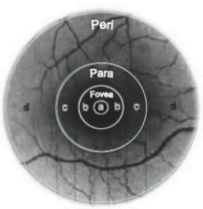
BINARY ROBUST INVARIANT SCALABLE KEYPOINTS (BRISK)*

- Keypoint detector based on FAST
- both rotation and scale invariant
- binary descriptor formed by pairwise intensity comparisons (like BRIEF) but on a radially symmetric sampling pattern
  - red circles: size of the smoothing kernel applied
  - blue circles: smoothed pixel value used
- 10× faster than SURF, slower than BRIEF

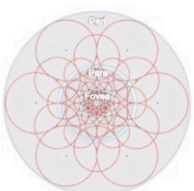


FAST RETINA KEYPOINT (FREAK)*

- rotation and scale invariant
- sampling pattern similar to BRISK but uses a more pronounced "retinal" (ie log-polar) sampling pattern inspired by the human retina: higher density at points near the center
- binary descriptor using pairwise comparison as in BRISK.
- pairs are learned (as in ORB).
- circles indicate size of smoothing kernel
- coarse-to-fine matching (cascaded approach)
  - first compare first half of bits
  - if distance smaller than threshold proceed to compare next bits etc.
- faster to compute, less memory than SIFT, SURF or BRISK.

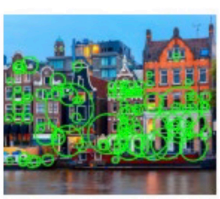


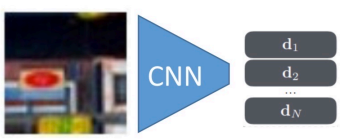
Human retina


FREAK sampling pattern

LEARNED INVARIANT FEATURE TRANSFORM (LIFT)*

- learning-based descriptor
- rotation, scale, viewpoint and illumination invariant
- First a network predicts the patch orientation which is used to derotate the patch
- Then another NN is used to generate a patch descriptor ($\in \mathbb{R}^{128}$)
- Illumination invariance is achieved by randomizing illuminations during training
- LIFT > SIFT in repeatability.

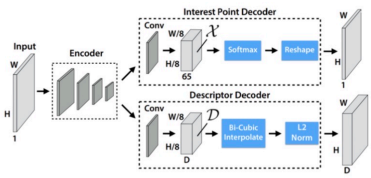

Keypoints with scales and orientations



neural network predicts descriptor

SELF-SUPERVISED INTEREST POINT DETECTION AND DESCRIPTION (SUPERPOINT)

- Joint regression of keypoint location and descriptor.
- Self-supervised
- Trained on synthetic images and refined on homographies of real images
- Detector less accurate than SIFT and LIFT but descriptor outperforms SIFT and LIFT.
- But slower than SIFT and LIFT.



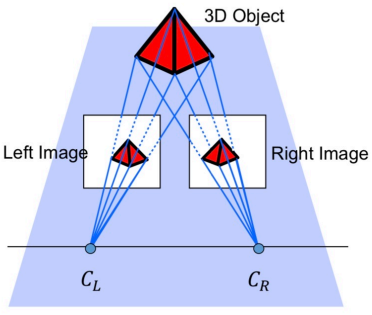
RECAP TABLE

Detector	Localization Accuracy of the detector	Descriptor that can be used	Efficiency	Relocalization & Loop closing
Harris	++++	Patch SIFT/LIFT BRIEF ORB BRISK FREAK	+++ + ++++ ++++ +++ ++++	+ +++++ +++ ++++ +++ ++++
Shi-Tomasi	++++	Patch SIFT BRIEF ORB BRISK FREAK	++ + ++++ ++++ +++ ++++	+ +++++ +++ ++++ +++ ++++
FAST	++++	Patch SIFT/LIFT BRIEF ORB BRISK FREAK	++++ + ++++ ++++ +++ ++++	+ +++++ +++ ++++ +++ ++++
SIFT	+++	SIFT	+	++++
SURF	+++	SURF	++	++++
SuperPoint	++	SuperPoint	+	+++++

STEREO VISION

Goal : recover the 3D structure by computing the intersection of corresponding rays.

Tasks :
- Multiple View Geometry
  - 3D reconstruction (K,T,R known)
  - Structure from Motion (K,T,R unknown)
- 2 View Geometry
  - Depth from stereo (K,T,R known)
  - 2 view Structure from Motion (K,T,R unknown)

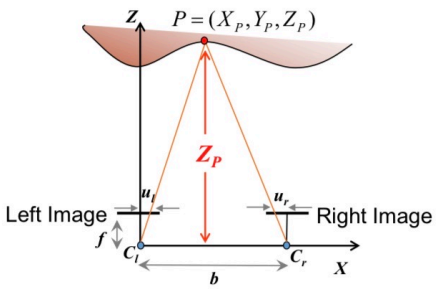


STEREO VISION

- **Goal**: find an expression of the 3D point coordinates as a function of the 2D image coords.
- **Assumptions**:
  - cameras are calibrated: both intrinsics and extrinsics are known
  - point correspondences are given

STEREO VISION - SIMPLIFIED CASE

- cameras are identical (same intrinsics);
- cameras are aligned on the x-axis.



Baseline = distance between the optical centers of the two cameras

From Similar Triangles:

$$\frac{f}{Z_P}=\frac{u_l}{X_P} \qquad \frac{f}{Z_P}=\frac{-u_r}{b-X_P} \qquad \Rightarrow \qquad Z_P=\frac{bf}{u_l-u_r}$$

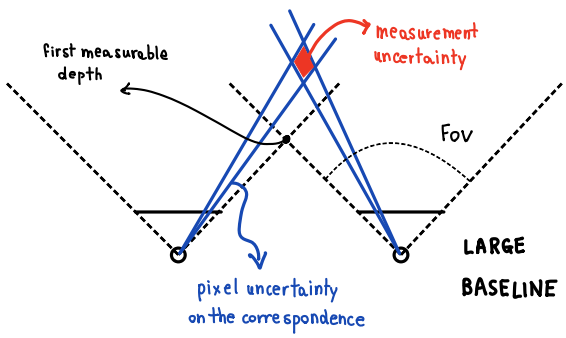
Disparity
difference in image location of the projection of a 3D point on two image planes

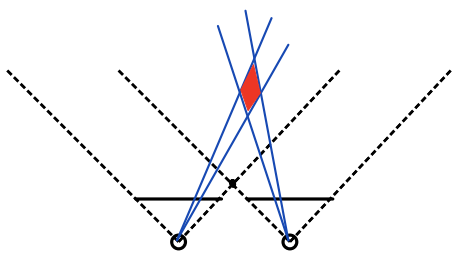
1. What's the max disparity of a stereo camera?
2. What's the disparity of a point at infinity?

LARGE VS SMALL BASELINE

- **Large baseline**
  - small depth error
  - minimum measurable depth increases
    ↳ difficult for close objects

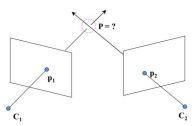
- **Small baseline**
  - Large depth error
  - minimum measurable depth increases
    ↳ easier for close objects.





STEREO VISION - GENERAL CASE

- Very hard to have perfectly aligned cameras
- Compute extrinsics and intrinsics with traditional methods (Tsai's or Zhang's method)
- **Triangulation**: problem of determining the 3D position of a point given a set of corresponding image locations and known camera points.
- In other words, we want to find the intersection of 2 rays but, because of noise and numerical errors, they can't meet exactly, so we need to compute an approximation.



$- a_z(a_x a_y) + a_y(a_z a_x)$
$= 0$

TRIANGULATION : LEAST SQUARE APPROXIMATION

- Left camera (here) assumed as world frame

$$\lambda_1 \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \lambda_1 p_1 = K_1 [I \mid 0] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = M_1 \cdot P \implies p_1 \times \lambda_1 p_1 = p_1 \times M_1 \cdot P = [p_1]_x \cdot M_1 \cdot P = 0$$

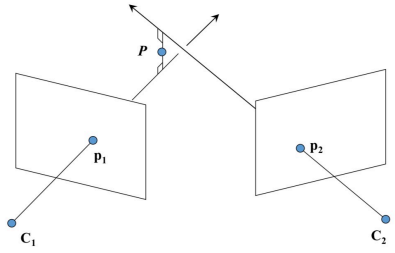
$$\lambda_2 \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \lambda_2 p_2 = K_2 [R \mid T] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = M_2 \cdot P \implies p_2 \times \lambda_2 p_2 = p_2 \times M_2 \cdot P = [p_2]_x \cdot M_2 \cdot P = 0$$

cross product

skew matrix $[a]_x = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$ rank = 2

homogeneous sys. of eqns. P determined via SVD, as for the DLT method.

- geometric interpretation : P is computed as midpoint of the shortest segment connecting the two lines.



TRIANGULATION : NON LINEAR REFINEMENT

- Initialize P using least square approximation
- refine P by minimizing sum of left and right squared reprojection error

$$P = \arg\min_P \| p_1 - \pi(P, K_1, I, 0) \|^2 + \| p_2 - \pi(P, K_2, R, T) \|^2$$

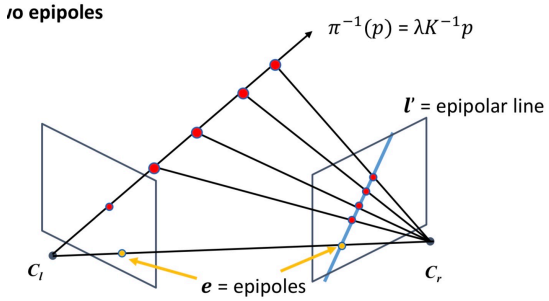
- can be minimized using Levenberg-Marquardt (more robust than Gauss-Newton to local minima)

CORRESPONDENCE PROBLEM

Correspondence Problem : given a point p_L on the left image, find correspondence p_R on right image.

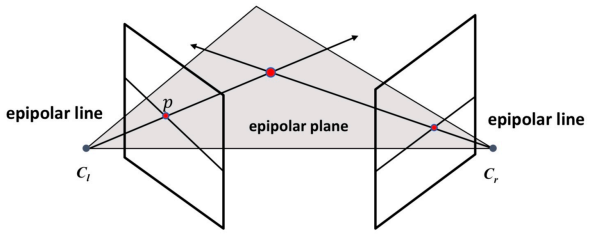
Search Problem : 2D exhaustive search is very expensive (N^2 comparisons for N descriptors).
We can use geometrical consistency checks

- potential matches for p have to lie on the corresponding epipolar line l'.
- the epipolar line is the back-projection of the ray $\pi^{-1}(p)$ onto the camera image.
- the epipole is the projection of the optical center on the other camera image.
- a stereo camera has two epipoles.



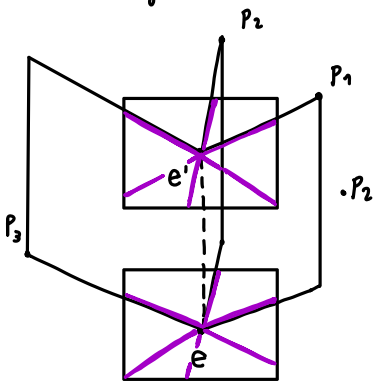
THE EPIPOLAR CONSTRAINT

- Camera centers C_l, C_r and the image point p define the epipolar plane.
- The intersection of the epipolar plane with the two image planes are called epipolar lines.
- Epipolar constraint: Corresponding points must lie along the epipolar line → makes the correspondence problem a 1D search.



Example: Forward Motion

- two consecutive image from the same camera
- epipole has the same coordinate in both images
- epipolar lines "radiating" from the epipole (focus expansion)

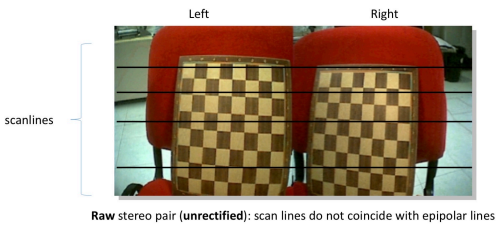


—: epipolar lines

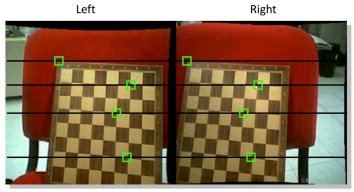
↑ camera moved forward

STEREO RECTIFICATION

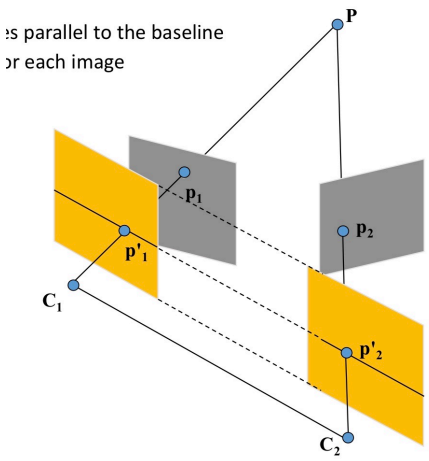
- no camera is perfectly calibrated
- more computationally efficient if scan-lines == epipolar lines
- Stereo rectification: wraps the left and right images into new "rectified" images such that the epipolar lines coincides with scanlines.
  - It works by computing 2 homographies which achieve the result



Raw stereo pair (**unrectified**): scan lines do not coincide with epipolar lines



Rectified stereo pair: **scanlines coincide with epipolar lines**



STEREO RECTIFICATION ALGORITHM

- The projection equation is $\lambda p = k[R|T]P$ where $R = R_{cw}$ and $T = T_{cw}$
- We use R_{wc} and T_{wc} as T_{wc} already gives us the camera position in world frame. Then the projection equation is $\lambda p = KR^{-1}(P-T) = kR^{-1}(P-C)$
- The goal of stereo rectification is to warp camera images such that they are coplanar (horizontal epipolar lines), so they have same $\hat{R}$ and same intrinsics (vertically aligned) so they have same $\hat{K}$

$$\lambda_L \begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix} = K_L R_L^{-1} \left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} - C_L \right) \quad \text{OLD LEFT CAMERA}$$

$$\lambda_R \begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix} = K_R R_R^{-1} \left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} - C_R \right) \quad \text{OLD RIGHT CAMERA}$$

$$\hat{\lambda}_L \begin{bmatrix} \hat{u}_L \\ \hat{v}_L \\ 1 \end{bmatrix} = \hat{K}\hat{R}^{-1} \underbrace{\left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} - C_L \right)}_{\lambda_L R_L K_L^{-1} \begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix}} \quad \text{NEW LEFT CAMERA}$$

$$\hat{\lambda}_R \begin{bmatrix} \hat{u}_R \\ \hat{v}_R \\ 1 \end{bmatrix} = \hat{K}\hat{R}^{-1} \underbrace{\left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} - C_R \right)}_{\lambda_L R_R K_L^{-1} \begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix}} \quad \text{NEW RIGHT CAMERA}$$

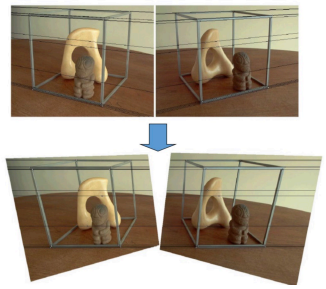
$$\Rightarrow \hat{\lambda}_L \begin{bmatrix} \hat{u}_L \\ \hat{v}_L \\ 1 \end{bmatrix} = \underbrace{\lambda_L \hat{K}\hat{R}^{-1} R_L K_L^{-1}}_{\text{homography of left camera}} \begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix} \qquad \hat{\lambda}_R \begin{bmatrix} \hat{u}_R \\ \hat{v}_R \\ 1 \end{bmatrix} = \underbrace{\lambda_L \hat{K}\hat{R}^{-1} R_R K_L^{-1}}_{\text{homography of right camera}} \begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix}$$

- How do we choose $\hat{K}$ and $\hat{R}$?
- Good choices are

$$\hat{K} = \frac{K_L + K_R}{2} \qquad \hat{R} = [\hat{r}_1, \hat{r}_2, \hat{r}_3] \quad \text{with} \quad \hat{r}_1 = \frac{C_R - C_L}{\|C_R - C_L\|} \quad : \text{new image planes are parallel to baseline}$$

$$\hat{r}_2 = r_3 \times \hat{r}_1 \quad : \text{where } r_3 \text{ is } 3^{rd} \text{ column of } R_L$$

$$\hat{r}_3 = \hat{r}_1 \times \hat{r}_2$$



Recap of the method
- First compute lens distortion
- then compute homographies and rectify
- use bilinear interpolation for warping.

DENSE STEREO CORRESPONDENCE : DISPARITY MAP

1. Rectify stereo pair (if not already rectified)
2. ∀ pixel in the left image, find its corresponding point in the right image.
3. Compute the disparity for each found pair of correspondences
4. Visualize it as a grayscale or color-coded image: disparity map

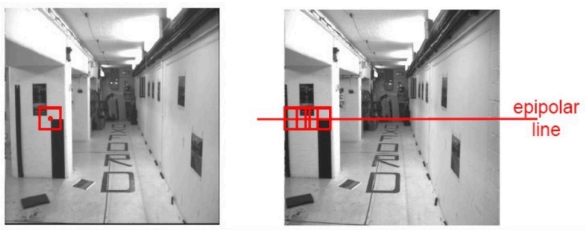
Left image | Right image

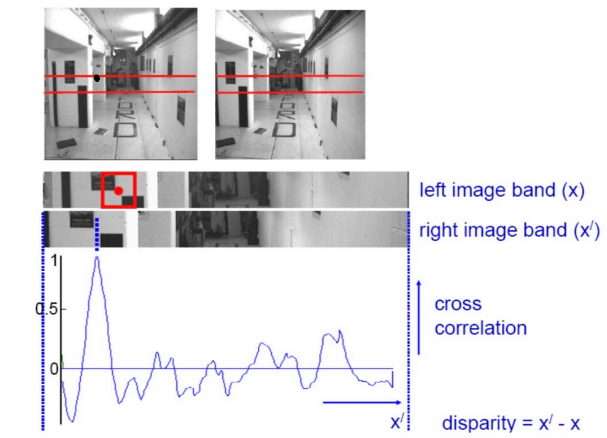
Close objects experience bigger disparity → appear brighter in disparity map

(5.) Once the stereo pair is rectified, the depth of each point can be computed recalling that $Z_P = b \cdot f / \text{disparity}$

CORRESPONDENCE PROBLEM - CONTINUED

- Once rectified correspondence search can be done along the same scanline
- To average noise effects, use a window around point of interest with the assumption that neighbouring pixels have similar intensity
- Find correspondence minimizing metric ((z)NCC, (z)SSD, (z)SAD, Census Transform ...)
- Texturless regions are not distinctive: high ambiguity for matches ↦ increase window size
  - Smaller window
    - ✗ more noise
    - ✓ more detail
  - larger window
    - ✗ smoother disparity map
    - ✓ less detail





CHALLENGES

- multiple possible matches for the same epipolar line
- occlusions and repetitive patterns
- non-lambertian surfaces (specularities) and texturless surfaces

SEMI-GLOBAL MATCHING (SGM)

- Popular open-source algo that computes dense disparity maps from a rectified stereo pair
- main idea: perform coarse to fine block matching followed by regularization (smoothing)
- the estimated disparity map is a piece-wide smooth surface passing through the initial disparity map

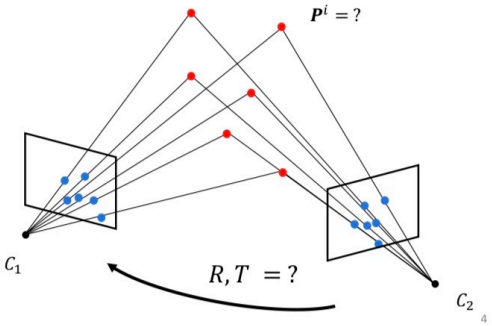
Left Image | Right Image | Estimated Disparity

STRUCTURE FROM MOTION (SFM)

Problem formulation: given a set of n point correspondences between two images $\{p_1^i = (u_1^i, v_1^i),\ p_2^i = (u_2^i, v_2^i)\}$ where $i = 1, \dots, n$ the goal is to simultaneously

- estimate the 3D points P_i
- the camera relative motion parameters (R, T)
- and the camera intrinsics that satisfy

$$\lambda_1 \begin{bmatrix} u_1^i \\ v_1^i \\ 1 \end{bmatrix} = K_1 [I|0] \begin{bmatrix} X_w^i \\ Y_w^i \\ Z_w^i \\ 1 \end{bmatrix} \qquad \lambda_2 \begin{bmatrix} u_2^i \\ v_2^i \\ 1 \end{bmatrix} = K_2 [R|T] \begin{bmatrix} X_w^i \\ Y_w^i \\ Z_w^i \\ 1 \end{bmatrix}$$



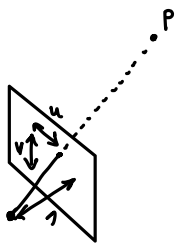
CALIBRATED CASE

- K_1, K_2 known
- We use $\bar{p} = K^{-1} p$ for convenience

$$\lambda_1^i \begin{bmatrix} \bar{u}_1^i \\ \bar{v}_1^i \\ 1 \end{bmatrix} = [I|0] P \qquad \lambda_2^i \begin{bmatrix} \bar{u}_2^i \\ \bar{v}_2^i \\ 1 \end{bmatrix} = [R|T] P$$

$\bar{P}_1$ $\qquad$ $\bar{P}_2$

↳ Vector in space with same direction as ray with $f = 1$



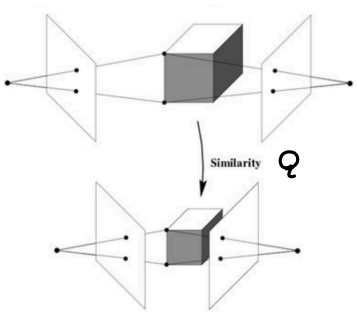
SCALE AMBIGUITY

- If we rescale the entire scene and camera views by a constant factor, the projections (in pixels) of the scene points in both images remain exactly the same.

$$p_1 = K_1 T_1 P \quad T_1 = [R_1|T_1]$$
$$p_2 = K_2 T_2 P$$

Given an affine scene transformation Q and its inverse Q^{-1}

$$p_1 = K_1 T_1 Q^{-1} Q P = K_1 \tilde{T}_1 \tilde{P}$$
$$p_2 = K_2 T_1 Q^{-1} Q P = K_2 \tilde{T}_2 \tilde{P}$$

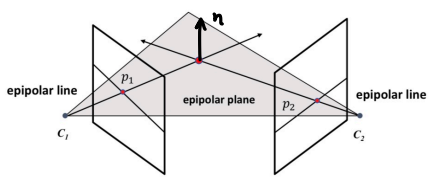


- In SFM it is therefore not possible to recover the absolute scale of the scene.
- Thus only 5 degrees of freedom are measurable: Rotation (3 dofs) and translation direction up to scale (2 dofs)

→ given n correspondences
- $4n$ knowns (correspondences equations $\lambda p = \Pi P$)
- $5 + 3n$ unknowns

→ $4n \geq 5 + 3n \Rightarrow \boxed{n \geq 5}$

- Can we estimate camera extrinsics without having to find the 3D position of points? Yes.
- How? We can use the epipolar geometry: the two rays and the baseline must be co-planar



$\bar{p}_1 = \begin{bmatrix} \bar{u}_1 \\ \bar{v}_1 \\ 1 \end{bmatrix}$ $\bar{p}_2 = \begin{bmatrix} \bar{u}_2 \\ \bar{v}_2 \\ 1 \end{bmatrix}$ $T = c_2 - c_1$ $\bar{p}_1, \bar{p}_2$ and T are coplanar

$\bar{p}_1$: normalized point in camera 1 reference frame
$\bar{p}_2$: normalized point in camera 2 reference frame

The extrinsics of $\bar{p}_2$ has $R_{c_2 w} = R_{c_2 c_1}$, $T_{c_2 w} = T_{c_2 c_1}$ since camera 1 is our world frame.
We can use these transformations to express $\bar{p}_1$ in c_2 frame and compute the normal to the plane:

$$^{c_2}\bar{p}_1 = R\bar{p}_1 + T$$
$$n = T \times (R\bar{p}_1 + T) = [T]_x R\bar{p}_1$$

And since $\bar{p}_2$ belongs to this plane:

$$\boxed{\bar{p}_2^T [T]_x R \bar{p}_1 = 0}$$ Epipolar Constraint

$$\boxed{E = [T]_x R}$$ Essential Matrix

Note that $\text{rank}(E) = \text{rank}(ER^T) = \text{rank}([T]_x)$ (rotation doesn't change rank) $= 2$

SOME HISTORY

- Kruppa showed in 1913 that 5 images correspondences is the minimal case and that there can be up to 11 solutions
- In 1988, Demazure shows that there are at most 10 distinct solutions.
- In 1996, Philipp proposed an iterative algo to find these solutions
- In 2004, Niester proposed the first efficient and non iterative solution.
- The first popular solution uses the 8-point algorithm or Longuet-Higgins algo.

THE 8 POINT ALGORITHM

Each pair of point correspondences $\bar{p}_1$ and $\bar{p}_2$ provide a linear equation $\bar{p}_2^T E \bar{p}_1$

$$\bar{u}_2\bar{u}_1 e_{11} + \bar{u}_2\bar{v}_1 e_{12} + \bar{u}_2 e_{13} + \bar{v}_2\bar{u}_1 e_{21} + \bar{v}_2\bar{v}_1 e_{22} + \bar{v}_2 e_{23} + \bar{u}_1 e_{31} + \bar{v}_1 e_{32} + e_{33} = \underbrace{[\cdots]}_{q^i} \begin{bmatrix} e_{11} \\ e_{12} \\ e_{13} \\ \vdots \\ e_{33} \end{bmatrix}$$ e_{ij} : element of E

For n correspondences q^i : $\begin{bmatrix} q^0 \\ q^1 \\ \vdots \\ q^n \end{bmatrix} \begin{bmatrix} e_{11} \\ \vdots \\ e_{33} \end{bmatrix} = Q \cdot \bar{E}$ ($\bar{E}$: unknown, Q: known)

- Minimum solution: Q must have rank 8 for a unique (non trivial) solution $\bar{E}$. As each point (if not coplanar) provides an independent equation, we need more than $n \geq 8$ points.

- An overdetermined solution is to minimize $\|Q\bar{E}\|^2$ subject to $\|E\|^2 > 1$ (avoid to select $\bar{E} = 0$)
- The solution is the eigenvector corresponding to the smallest eigenvalue of the matrix Q^TQ since it minimizes $\|Qx\| = x^T Q^T Q x = x^T \lambda_{min} x = \lambda_{min} x^T x = \lambda_{min} \underbrace{\|x\|}_{=1} = \lambda_{min}$
- It can be solved via SVD
  $Q = U S V^T$ $Q^TQ = V S U^T U S V = V S^2 V$
  → $[U,S,V] = \text{svd}(Q)$ $x = V[:,9]$ (last eigenvector) → $E = \text{reshape}(x,3,3)$

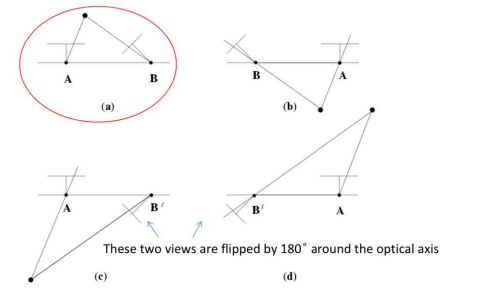
EXTRACT R AND T

- E has rank 2 : $\text{rank}(E) = \text{rank}([T]_\times R) = \text{rank}([T]_\times) = 2$
- Enforce rank = 2 setting the minimum eigenvalue to 0 : $E = U\Sigma V^T = U \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \cancel{\sigma_3} \end{bmatrix} V^T$

$$\hat{T} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \Sigma V^T \qquad \hat{T} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & t_x \\ -t_y & t_x & 0 \end{bmatrix} \Rightarrow \hat{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

$$\hat{R} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T \qquad T = K_2 \hat{t} \qquad R = K_2 \hat{R} K_1^{-1}$$

- There are 4 possible solutions but only one solution where points are in front of both cameras.



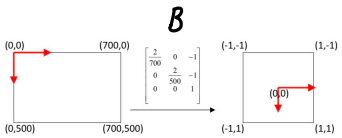
STRUCTURE FROM MOTION (UNCALIBRATED CASE)

- So far we have used normalized pixel coordinates $\bar{p} = K^{-1} p$
- Then $\bar{p}_2^T E \bar{p}_1 = p_2^T K_2^{-T} E K_1^{-1} p_1 = 0$
- The same algorithm holds and we call $F = K_2^{-T} E K_1^{-1}$ the Fundamental Matrix.
- The advantage is that we work directly in pixel coordinates.

PROBLEMS WITH 8-POINT ALGORITHM

- Poor numerical conditioning which makes results very sensitive to noise.
- Can be fixed by rescaling the data : normalized 8 point algorithm.

NORMALIZED 8-POINT ALGORITHM



- Transform image coordinates so that they are in the range $[-1,1] \times [-1,1]$
  $\hat{p} = Bp$
- Estimate normalized $\hat{F}$ from $\hat{p}_1$ and $\hat{p}_2$
- Compute unnormalized F from $\hat{F}$

$$\hat{p}_2^T \hat{F} \hat{p}_1 = 0 \rightarrow p_2^T \underbrace{B_2^T \hat{F} B_1}_{F} p_1 \rightarrow F = B_2^T \hat{F} B_1$$

- In the original paper it was proposed to rescale each points sets such that each has centroid 0 and the standard deviation $\sqrt{2}$: $\hat{p}^i = \sqrt{2}/\delta \, (p^i - \mu)$; $\mu = \Sigma p^i / N$ $\quad \delta = 1/N \, \Sigma_{i=1}^{N} \| p^i - \mu \|^2$
- This transformation can be expressed in matrix form using homogeneous coordinates

$$\hat{p}_i = \begin{bmatrix} \sqrt{2}/\delta & 0 & -\frac{\sqrt{2}}{\delta}\mu_x \\ 0 & \sqrt{2}/\delta & -\frac{\sqrt{2}}{\delta}\mu_y \\ 0 & 0 & 1 \end{bmatrix} p^i$$

- Can we extract R, T, K_1, K_2 from F?
  In general no: infinite solutions exists.
- However if the coordinates of the principal points of each camera are known and the two camera have the same focal length f in pixels, then R, T and f can be determined uniquely

ERROR MEASURES

The quality of the estimated Essential or Fundamental matrix can be measured using different error metrics:

- Algebraic error
- Directional error
- Epipolar line distance
- Reprojection error

} can only be $\emptyset$ for 8 points (or more and no noise)

Algebraic error: $err = \|Q\bar{E}\|^2 = \sum_{i=1}^{N} (\bar{p}_2^{i\,T} E \bar{p}_1^{i})$

Also $\|\bar{p}_2^{i\,T} E \bar{p}_1^{i}\| = \|\bar{p}_2^{T} \cdot (E\bar{p}_1)\| = \|\bar{p}_2\| \|E\bar{p}_1\| \cos(\theta)$ ($n \sim E\bar{p}_1$; θ: angle between $\bar{p}_2$ and normal)

$\Rightarrow$ if this is non-zero $\bar{p}_1, \bar{p}_2$ and n are not coplanar (but depends also on norms $\|p_i\|$!)

Directional error: cosine of the angle from the epipolar plane $err = \sum_{i=1}^{N} \cos(\theta_i)^2$ (does not depend on norms)

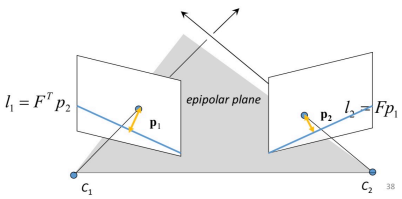
It is obtained normalizing the algebraic error: $\cos(\theta) = \bar{p}_2^{T} E \bar{p}_1 \,/\, \|\bar{p}_2^{T} E \bar{p}_1\|$

Epipolar line distance: sum of squared epipolar line to point distances $\rightarrow err = \sum_{i=1}^{N} d(p_1^i, l_1^i)^2 + d(p_2^i, l_2^i)^2$

Cheaper than reprojection error because it does not require point triangulation

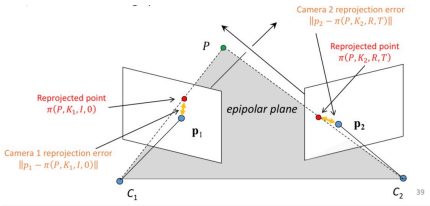
$p_2^T F p_1$ can be read as

- $p_2^T e_2$ with e the coefficients of epipolar line in the second image
- $p_1^T F^T p_2 = p_1^T e_1$ with $e_1 = F^T p_2$ the coefficients of the epipolar line in the first image



Reprojection error: sum of the squared Reprojection errors $\rightarrow err = \sum_{i=1}^{N} \|p_1^i - \pi(P^i, K_1, I, 0)\|^2 + \|p_2^i - \pi(P^i, K_2, R, T)\|^2$

- More expensive to evaluate then previous because requires triangulation of points P^i in 3D!
- However most popular because it is the most accurate



ROBUST ESTIMATION

Often matches contain outliers:
- repetitive features
- changes in view point (scale also) and illumination
- image noise
- occlusions
- moving objects / blur

EXPECTATION MAXIMIZATION

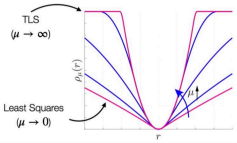
Method for model fitting in presence of outliers.

Example: line fitting
1. Estimate line parameters using least squares: $\min \Sigma r_i^2$ (r_i = point line distance)
2. Calculate new r_i $\forall$ point and assign each a weight (e.g. $e^{-r_i^2}$ representing the likelihood that such an assignment is correct. (estimates the expectation)
3. Re-estimate line parameters (e.g using weighted least squares: $\min \Sigma w_i r_i^2$). (Maximization Step)
4. Iterate 2-3 until convergence
5. Select the inliers as data points with weight higher than threshold.

Problems
- Very sensitive to the initial condition:
  - EM selects the initial condition minimizing sum of residuals
  - While this is a convex function, the result is influenced by a few large error values (distant outliers)
  - EM converges to the wrong solution if the initial condition is too far from the true one
- Alternatives
  - GNC algorithm
  - RANSAC algorithm

GRADUATED NON CONVEXITY (GNC) ALGORITHM

- Idea: optimize a surrogate function $\Sigma \rho_\mu(r_i)$ where μ controls the amount of non-convexity.
  - Start by solving the non robust convex least square ($\mu \to 0$)
  - At each iteration gradually increase non convexity ($\mu \to \infty$) and recompute w_i
  - It is shown to be robust up to 90% of outliers with five times fewer iterations than RANSAC.
  - However RANSAC can cope with more than 90% outliers



RANSAC (RAndom SAmple Consensous)

- RANSAC is the standard method for model fitting in the presence of outliers
- It is non deterministic: you get a different result every time you run it. (in contrary to EM and GNC)
- It is non sensitive to the initial condition and does not get stuck in local maxima

Example: line fitting

1. Select a sample of 2 points at random
2. Select the model parameters from the sample
3. Calculate residual ∀ data point
4. Select data that support current hypothesis
5. Repeat 1-4 K times
6. Select the set with maximum number of inliers obtained with K iterations.

HOW MANY ITERATIONS?

- w := number of inliers/N (probability of selecting an inlier)
- N := total number of data points
- Assuming that 2 points necessary to find a line:
  - w^2 = P(both selected points are inliers)
  - $1-w^2$ = P(at least one of this point is an outlier)
- Let k be the number of RANSAC iterations executed so far
  - $(1-w^2)^k$ = P(RANSAC never selected two points that are both inliers)
- p := probability of success

  $1-p = (1-w^2)^k$ and therefore

$$\boxed{k = \frac{\log(1-p)}{\log(1-w^2)}}$$

- In practice we only need a rough estimate of w
- More advance tecniques estimate w and adaptively update it at every iteration.

RANSAC FOR SFM

1. What is the model in SFM?
   - The Essential or Fundamental matrix
   - Alternatively R and T

2. What's the minimum number of points to estimate the model?
   - Theoretically 5 for calibrated cameras
   - 8 for the 8-point algo (both calibrated and not)

3. How do we compute the distance of a point from the model?
   - Algebraic error
   - Directional error
   - ...

RANSAC ITERATIONS K vs S.

Assume $p = 99\%$ and $\epsilon = 1-w = 50\%$ (Fraction of outliers)

- 8 point RANSAC → K = 1177 Good method but too many iterations
- 5 point RANSAC → K = 145 Efficient but can return up to 10 solutions of E
- 2 point RANSAC (e.g line fitting) → K = 16

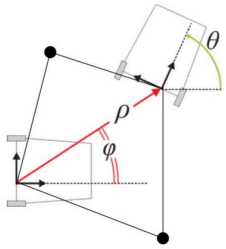
MOTION CONSTRAINTS

We can use motion constraints to reduce the solution space.

Planar Motion

Planar motion is described by three parameters: ϑ, φ, ρ

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\varphi \\ \rho\sin\varphi \\ 0 \end{bmatrix}$$



Observe that E has 2DoF (θ, φ, because ρ is the scale factor); thus, **2 correspondences are sufficient** to estimate θ and φ [Ortin, 2001]

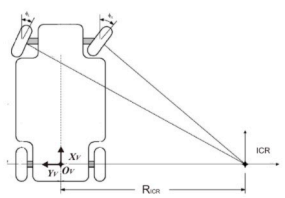
$$E = [T_\times]R = \begin{bmatrix} 0 & 0 & \rho\sin(\varphi) \\ 0 & 0 & -\rho\cos(\varphi) \\ -\rho\sin(\varphi-\theta) & \rho\cos(\varphi-\theta) & 0 \end{bmatrix}$$

"2-Point RANSAC", Ortin & Montiel, Indoor robot motion based on monocular images, Robotica, 2001. PDF.

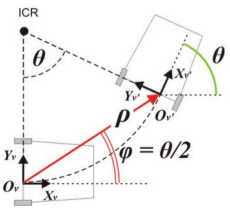
49

We can use less than 2 correspondences if we exploit wheeled vehicles with non-holonomic constraints.

- Wheeled vehicles like cars follow locally planar circular motion about the Instantaneous Center of Rotation (ICR)
- $\varphi = \theta/2$ → only 1 DoF, thus only 1 correspondence is needed
- This is the smallest parametrization possible



Example of Ackerman steering principle



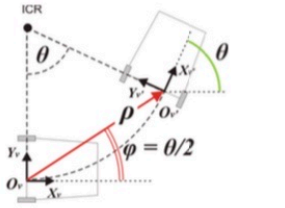
Locally-planar circular motion

Let's compute the Epipolar Geometry

$E = [T_\times]R$ *Essential matrix*

$\bar{p}_2^T E \bar{p}_1 = 0$ *Epipolar constraint*

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \rho\cos\frac{\theta}{2} \\ \rho\sin\frac{\theta}{2} \\ 0 \end{bmatrix}$$



Locally-planar circular motion

$$E = [T_\times]R = \begin{bmatrix} 0 & 0 & \rho\sin\frac{\theta}{2} \\ 0 & 0 & -\rho\cos\frac{\theta}{2} \\ -\rho\sin\frac{\theta}{2} & \rho\cos\frac{\theta}{2} & 0 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \rho\sin\frac{\theta}{2} \\ 0 & 0 & \rho\cos\frac{\theta}{2} \\ \rho\sin\frac{\theta}{2} & -\rho\cos\frac{\theta}{2} & 0 \end{bmatrix}$$

Notice that ρ can be cancelled out

$$p_2^T E p_1 = 0 \Rightarrow \sin\left(\frac{\theta}{2}\right)\cdot(u_2+u_1) + \cos\left(\frac{\theta}{2}\right)\cdot(v_2-v_1) = 0$$

$$\theta = -2\tan^{-1}\left(\frac{v_2-v_1}{u_2+u_1}\right)$$

Scaramuzza, 1-Point-RANSAC Structure from Motion for Vehicle-Mounted Cameras by Exploiting Non-Holonomic Constraints, International Journal of Computer Vision, 2011. PDF.

54

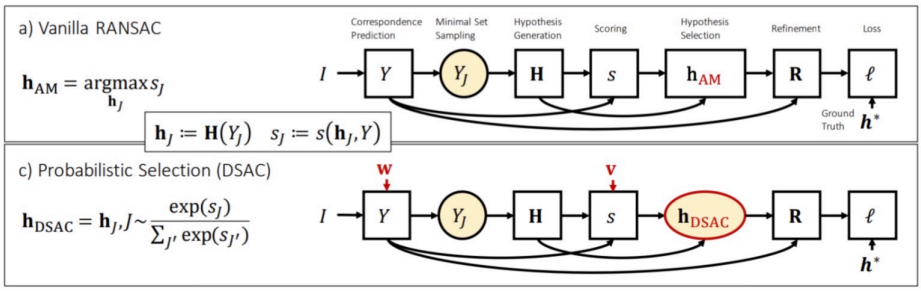
1-POINT RANSAC ALGORITHM

1. Compute θ ∀ point correspondence
2. Visualize histogram and take θ as the mean → only 1 iteration for removing outliers ($\theta' \notin [\bar{\theta} \pm \Delta\theta]$, threshold)
3. 1-Point RANSAC is only used to find inliers, motion is then estimated from them in 6 DoF.

DIFFERENTIABLE RANSAC

- RANSAC is not differentiable since it relies on selecting an hypothesis based on maximizing the number of inliers.
- DSAC introduces differentiable sample consensous

- Replaces max with softmax operator for model selection

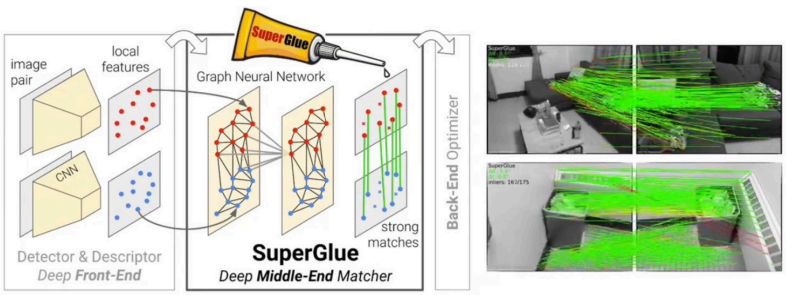


DEEP FUNDAMENTAL MATRIX ESTIMATION

- Input: two sets of noisy local features (coordinates + descriptors) contamined by outliers
- Output: fundamental matrix
- Idea: solve a weighted homogeneous LS problem where robust weights are estimated using deep networks.
- Robust: handles extreme wide-baseline image pairs.

SUPER GLUE: LEARNING FEATURE MATCHING WITH GRAPH NEURAL NETWORKS

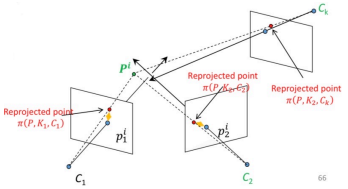
- Input: two sets of noisy local features (coordinates + descriptors) contamined by outliers
- Output: strong & outlier free matching
- Idea: combines DL with classical optimization
- Robust: handles extreme wide baseline pairs.



Sarlin, DeTone, Malisiewicz, Rabinovich, *SuperGlue: Learning Feature Matching with Graph Neural Networks*, International Conference on Computer Vision and Pattern Recognition (CVPR), 2020. PDF. Code.

2-VIEW BUNDLE ADJUSTMENT

- Non linear optimization of P^i and motion R,T
- Commonly used after LS (e.g 8-Point) estimation of R and T ($C_i = [R_i, T_i]$)
- Optimizes squared reprojection errors



$$P^i, C_2, \dots, C_n = \underset{P^i, C_{2\dots n}}{\arg\min} \sum_{k=1}^{\hat{N}} \sum_{i=1}^{N} \| p_k^i - \pi(P^i, K_k, C_k) \|^2$$

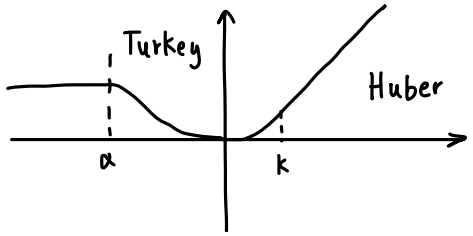
- when $n > 2$ is the n-view BA
- N.B: we assume $C_1 = [I, 0]$

- First camera R,T can be arbitrary
- Occasionally residual terms are weighted
- Initial values P^i, R, T should be close to optimum
- Can be minimized using Levenberg-Marquardt (more robust than Gauss-Newton to local minima)
- Can be modified to also optimize intrinsics parameters.

HUBER AND TURKEY NORMS

To prevent that large reprojection errors can negatively impact the optimization, a more robust norm $\rho(\cdot)$ is used instead of L_2.

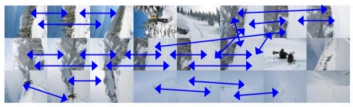
- Huber norm $\rho(x) = \begin{cases} x^2 & \text{if } |x| \leq k \\ k(2|x| - k) & \sim \end{cases}$
- Turkey norm $\rho(x) = \begin{cases} \alpha^2 & \text{if } |x| \geq \alpha \\ \alpha^2 \left(1 - \left(1 - \left(\frac{x}{\alpha}\right)^2\right)^3\right) & \sim \end{cases}$

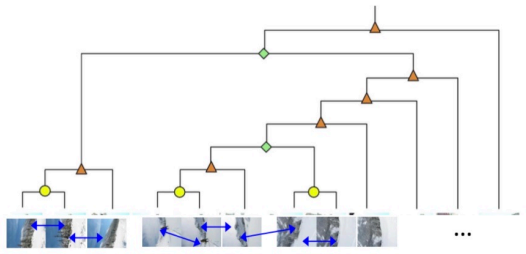


n-VIEW STRUCTURE FROM MOTION

HIERARCHICAL SFM

1. Extract and match features between nearby frames
2. Build cluster consisting of nearby frames
3. Extract topological tree (e.g count number of SIFT features matches)
4. Start from the terminal nodes
   - compute 2-view SFM and build 3D model (= point cloud)
5. Iterate according to the tree structure:
   - 5.1 Merge new view by running 3-Point RANSAC between 3D model and 3rd view
   - 5.2 Merge nearby models by running again 3 point RANSAC between one 3D model and one view of the other 3D model
   - 5.3 Bundle adjust





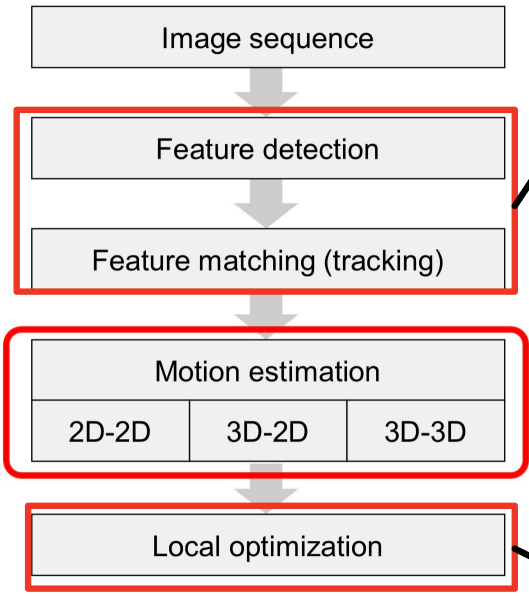
The circle ○ corresponds to the creation of a stereo-model, the triangle △ corresponds to applying PNP, the diamond ◇ corresponds to a fusion of two partial independent models.

SEQUENTIAL STRUCTURE FROM MOTION (also called VISUAL ODOMETRY)

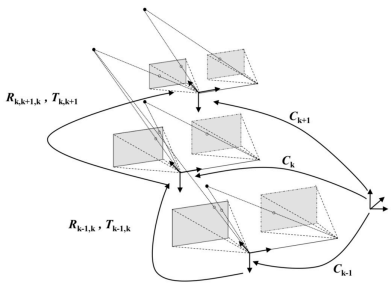
- Initialize structure from motion from 2 views (bootstrapping)
- For each additional view
  - Determine motion (pose) (localization)
  - Extend structure → can triangulate features during localization (mapping)
  - Refine structure and motion through bundle adjustment (optimization)

VO PIPELINE (RECAP)

VO computes the camera path incrementally (pose after pose)



Features tracked over multiple recent frames overlaid on the last frame



Example: Bundle Adjustment:

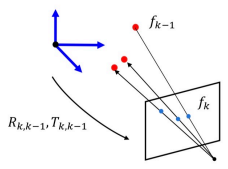
$$P^i, C_2, \dots, C_k = argmin_{P^i, C_2, \dots, C_k} \sum_{k=1}^{n} \sum_{i=1}^{N} \left\| p_k^i - \pi(P^i, K_k, C_k) \right\|^2$$

Or Pose-Graph Optimization (see later)

MOTION ESTIMATION

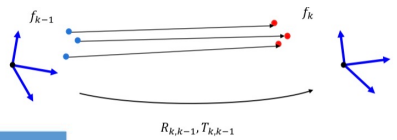
- Motion from 2D to 2D feature correspondences
  - both feature correspondences f_{k-1} and f_k are specified in image coordinates (2D)
  - the minimal case solution involves 5 features correspondences
  - popular algorithm 5 and 8-point algorithms

- Motion from 3D to 2D feature correspondences (i.e. Perspective from n points : PnP problem)
  - f_{k-1} is specified in 3D and f_k in 2D
  - DLT algorithm: minimal case is 6 points from 3D objects of 4 from planar objects
  - P3P algorithm: minimal case is 3 points (+1 for disambiguation)
  - EPNP algorithm: For more than 4 points



- Motion from 3D to 3D (also known as point cloud registration)
  - both f_{k-1} and f_k are specified in 3D. To do this it is necessary to triangulate the new features (need stereo camera)
  - the minimal case solution involves 3 non-collinear correspondences
  - popular algorithm consists of solving the system of equations with R and T unknown

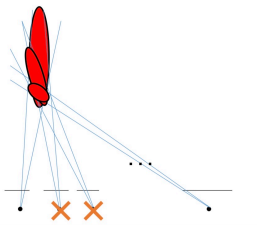
$$\begin{bmatrix} X_{k-1}^i \\ Y_{k-1}^i \\ Z_{k-1}^i \end{bmatrix} = [R|T] \begin{bmatrix} X_k^i \\ Y_k^i \\ Z_k^i \end{bmatrix}$$



Type of correspondences	Monocular	Stereo
2D-2D	X	
3D-2D	X	X
3D-3D		X

CASE STUDY: MONOCULAR VO

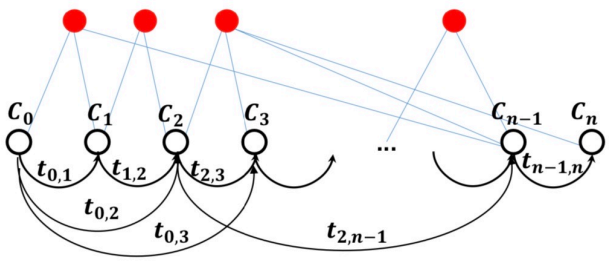
1. **Bootstrapping** (i.e. initialization)
   - Initialize structure from motion from 2 views, e.g. 5-8 point RANSAC. The first extracted relative pose of the second camera allows to triangulate the detected features and obtain a first 3D point cloud (structure) of features.
   - Refine structure and motion (Bundle Adjustment)
2. **Keyframe selection** (i.e skipping frames)
   - When frames are taken at nearby positions compared to the scene distance, 3D points will exibit large uncertainty
   - One way to avoid this consists of skipping frames until the average uncertainty of the 3D points normalized by the average distance of the scene, falls below a certain threshold. The selected frames are called Keyframes.
   - Rule of thumb: add a keyframe when Keyframe distance / average-depth > threshold (~ 10/20%)



3. **Localization** (i.e. pose estimation from a given point cloud)
   - Given a 3D point cloud (triangulated feature map) determine the pose of each additional view
   - This is a case of 3D to 2D correspondences
4. **Mapping** (i.e. extend structure)
   - Add the newly triangulate features to the point cloud
   - It is not necessary to do this at every frame since neighboring frames have almost the same features
   - More computation but can do loop-closure

LOCAL OPTIMIZATION

Refine structure and motion (often offline)



t_{iK} : transformation from camera pose C_K to camera pose C_i : $C_K t_{Ki}$

- Pose graph optimization $\{C_1, \dots, C_n\} = \arg\min_{\{C_1,\dots,C_n\}} \Sigma_i \Sigma_j \| C_i - C_j t_{ji} \|^2$
- Bundle adjustement : similar to pose-graph optimization but it also optimizes 3D points

$$P^i, C_1, \dots, C_n = \arg\min_{P^i, C_1, \dots, C_n} \sum_{K=1}^{k} \sum_{i=1}^{m} \rho(p_K^i - \pi(P^i, K_K, C_K))$$

$\rho(\cdot)$: Huber or Turkey norm

BUNDLE ADJUSTEMENT VS POSE-GRAPH OPTIMIZATION

- BA is more precise than pose graph optimization it adds additional constraints (landmark constraints)
- BA is more costly $O((qN+lm)^3)$ with N being the number of points, m the number of camera poses and q and l the number of parameters for points and camera poses → workarounds:
  - A small window size to make real time implementation possible
  - Not optimize over the 3D landmarks (motion only BA)

PLACE RECOGNITION

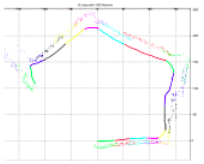
During VO two problems can occour:

1. Relocalization problem: feature tracking can be lost (occlusion, low texture, quick illumination change) or in case of monocular VO pure rotation followed by translation (multiple solutions exists?)
   → solution: relocalize camera pose and continue

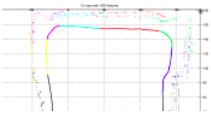
2. Loop closing problem: when you go back to a previously mapped area
   - Loop closure detection: to avoid map duplication
   - Loop correction: to compensate the accumulated drift

VO vs. Visual SLAM (recap from Lecture 01)

- **Visual Odometry**
  - Focus on incremental estimation
  - **Guarantees local consistency** (i.e., estimated trajectory is locally correct, but not globally, i.e. from the start to the end)
- **Visual SLAM** (Simultaneous Localization And Mapping)
  - **SLAM = visual odometry + loop detection & closure**
  - **Guarantees global consistency** (the estimated trajectory is globally correct, i.e. from the start to the end)



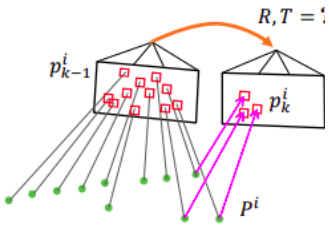
Visual odometry



Indirect vs Direct Methods

- **Indirect methods**
  1. Extract & match features + 3-point RANSAC
  2. Bundle Adjust by minimizing the **Reprojection Error**:

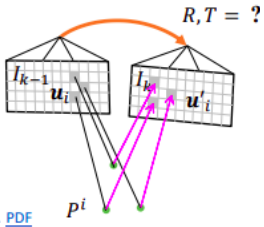
$$P^i, R, T = \arg\min_{P^i,R,T} \sum_{i=1}^{N} \rho\left(p_k^i - \pi(P^i, K, R, T)\right)$$



- ✓ Can cope with large frame-to-frame motions (large basin of convergence)
- × Slow due to costly feature extraction, matching, and outlier removal (e.g., RANSAC)

- **Direct methods**
  1. No feature extraction & no RANSAC needed. Instead, directly minimize **Photometric Error**:

$$P^i, R, T = \arg\min_{P^i,R,T} \sum_{i=1}^{N} \rho\left(I_{k-1}(p_{k-1}^i) - I_k\left(\pi(P^i, K, R, T)\right)\right)$$



- ✓ All information in the image can be exploited (higher accuracy, higher robustness to motion blur and weak texture (i.e., weak gradients))
- ✓ Increasing the camera frame-rate reduces computational cost per frame (no RANSAC needed)
- × Very sensitive to intial value → limited frame-to-frame motion (small basin of convergence)

What are their pros and cons?

Irani, Anandau, *All about direct methods*, Springer'99. PDF

Open Source Monocular VO and SLAM algorithms

- PTAM
- ORB-SLAM
- SVO
- LSD-SLAM
- DSO

Indirect methods: Minimize the feature reprojection error

Direct methods: Minimize the feature photometric error

PTAM: Parallel Tracking and Mapping

- Monocular only
- **Feature based**
  - FAST corners + patch descriptors
  - Minimizes reprojection error
  - **Jointly optimizes poses & structure** (sliding window BA)
- First to propose **keyframe-based VO**
- First to propose **alternation of localization** (i.e., camera tracking) and **mapping**
- Tracking and mapping running in **two independent threads**: updated map is used by localization thread asynchronously, as soon it becomes available
- Includes:
  - **Relocalization**
  - No global optimization, only local
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

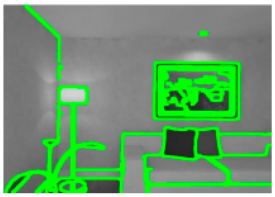
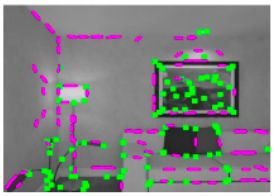
Klein, Murray, *Parallel Tracking and Mapping for Small AR Workspaces*, Internatic
PDF, code, videos. **Best pa**

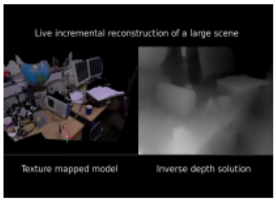
ORB-SLAM

- Supports both **monocular and stereo** cameras
- **Feature based**
  - FAST corners + ORB descriptors
  - ORB: binary descriptor, very fast to compute and match (Hamming distance)
  - **Jointly optimizes poses & structure** (sliding window BA)
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- Includes:
  - **Loop closing**
  - **Relocalization**
  - **Final optimization**
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

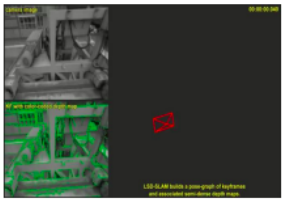
Mur-Artal, Montiel, Tardos, *ORB-SLAM: Large-scale Feature-based SLAM*, I

Direct Methods: Dense, Semi-dense, Sparse

Dense methods track every pixel	Semi-Dense methods track only edges	Sparse methods track sparse pixels
		
In a VGA image: 300'000+ pixels	In a VGA image: ~10,000 pixels	In a VGA image: ~2,000 pixels



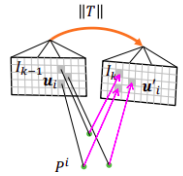
In a VGA image: 300'000+ pixels



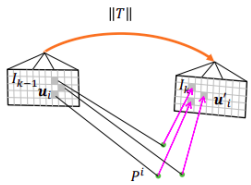
In a VGA image: ~10,000 pixels

In a VGA image: ~2,000 pixels
e.g., 120 feature patches × (4×4 pixels per patch)

DTAM



For **small motion** baselines, $\|T\|$, the **photometric error is usually small**

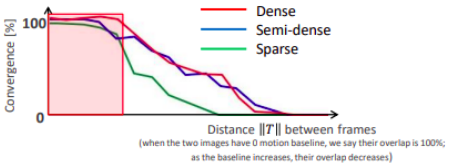


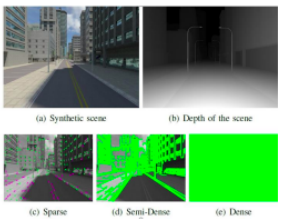
For **large motion** baselines, $\|T\|$, the **photometric error is usually large**

Direct Methods: Dense, Semi-dense, Sparse

- What is the influence of the motion baseline on the convergence rate of direct methods?

We can use **photorealistic simulation** to answer this question by generating thousands of data





Simulated dataset from here

- Findings:
  - **Dense and Semi-dense** behave similarly
  - **Weak gradients are not informative** for the optimization
  - **Dense methods** are only useful **with motion blur, defocus**, and **weak- texture** regions
  - **Sparse methods behave equally well as dense or semi-dense methods** for small motion baselines

LSD-SLAM Large Scale Semi Dense - SLAM

- Supports both **monocular and stereo** cameras
- **Direct** (photometric error) + **Semi-Dense** formulation
  - **3D structure** represented as **semi-dense** depth map
  - Minimizes **photometric error**
  - **Separateley** optimizes poses & structure (sliding window)
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- Includes:
  - **Loop closing**
  - **Relocalization**
  - **Final optimization**
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

DSO Direct Sparse Odometry

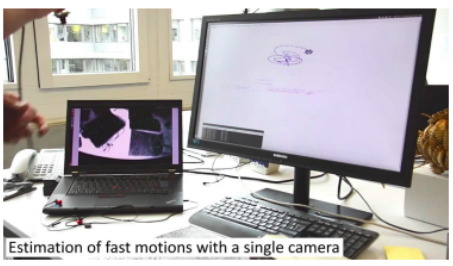
- Supports both **monocular and stereo** cameras
- **Direct** (photometric error) + **Sparse** formulation
  - **3D structure** represented as **sparse large gradients'** depth map
  - Minimizes **photometric error**
  - **Jointly optimizes poses & structure** (sliding window)
  - Incorporates photometric correction to compensate exposure time change ($\Delta t_{k-1}, \Delta t_k$)

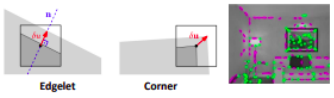
$$P^i, R, K = \arg\min_{P^i,R,K} \sum_{i=1}^{N} \rho\left(I_{k-1}(p^i_{k-1}) - \frac{\Delta t_{k-1}}{\Delta t_k} I_k\left(\pi(P^i, K, R, T)\right)\right)$$

- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

SVO Semi Direct Visual Odometry

- Supports both **monocular, stereo, multi-camera systems** as well as omnidirectional models (fisheye and catadioptric)
- Combines **indirect** + **direct methods**
  - **Direct** methods for **frame-to-frame motion estimation**
  - **Indirect** methods for **frame-to-keyframe pose refinement**
- **Mapping**
  - **Probabilistic depth** estimation (heavy-tail Gaussian distribution)
- Includes:
  - **Loop closing,**
  - **Relocalization,**
  - **Final optimization**
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- **Faster than real-time: up to 400 fps** on i7 laptops and **100 fps** on smartphone PCs (Odroid (ARM)) or NVIDIA Jetson


Estimation of fast motions with a single camera



	Mean	CPU@20 fps
SVO Mono	2.53	55 ±10%
ORB Mono SLAM (No loop closure)	29.81	187 ±32%
LSD Mono SLAM (No loop closure)	23.23	236 ±37%
DSO	20.12	181 ±27%

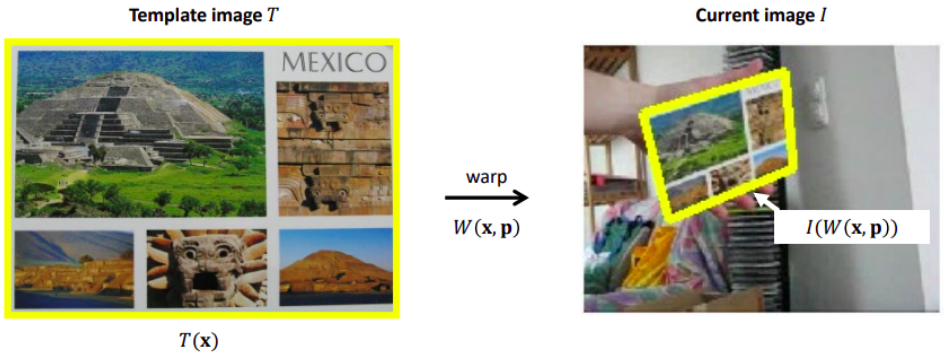
↑ Processing time in milliseconds ↑ CPU load (100% = 1 core)

Template tracking

Goal: **follow** a template image in a video sequence

Problem Formulation

Goal: estimate the transformation W (warp) between a template T and the current image



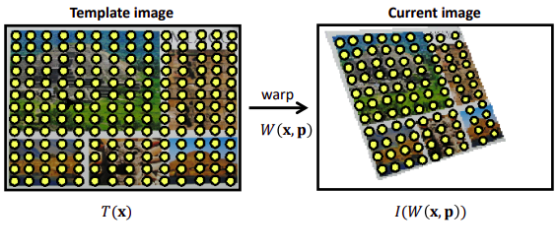
Two possible approaches

- Indirect methods
  - ✓ Can cope with large frame-to-frame motions (large basin of convergence)
  - × Slow due to costly feature extraction, matching, and outlier removal (e.g., RANSAC)
- Direct methods
  - ✓ All information in the image can be exploited (higher accuracy, higher robustness to motion blur and weak texture (i.e., weak gradients))
  - ✓ Increasing the camera frame-rate reduces computational cost per frame (no RANSAC needed)
  - × Very sensitive to intial value → limited frame-to-frame motion (small basin of convergence)

Direct methods work by directly processing pixel intensities

Goal: estimate the parameters **p** of the transformation $W(\mathbf{x}, \mathbf{p})$ that minimize the Sum of Squared Differences:

$$\mathbf{p} = argmin_{\mathbf{p}} \sum_{\mathbf{x} \in \mathbf{T}} [I(W(\mathbf{x}, \mathbf{p})) - T(\mathbf{x})]^2$$



* Every yellow dot in this image denotes a pixel

Indirect methods work by detecting and matching features

1. Detect and match features that are invariant to scale, rotation, view point changes (e.g., SIFT)
2. Geometric verification (RANSAC) (e.g., 4-point RANSAC for planar objects, or 5 or 8-point RANSAC for 3D objects)
3. Refine estimate by minimizing the sum of squared reprojection errors between the observed feature $\boldsymbol{f}^i$ in the current image and the warped corresponding feature $W(\mathbf{x}^i, \mathbf{p})$ from the template

$$\mathbf{p} = argmin_{\mathbf{p}} \sum_{i=1}^{N} [W(\mathbf{x}^i, \mathbf{p}) - \boldsymbol{f}^i]^2$$

- Pros: can cope with **large frame-to-frame motion** and **strong illumination changes**
- Cons: **computationally expensive**

Assumptions

- **Brightness constancy**

  The intensity of the pixels to track does not change much over consecutive frames → **It does not cope with strong illumination changes**

- **Temporal consistency**

  Small frame-to-frame motion (1-2 pixels).

  → **It does not cope with large frame-to-frame motion.** However, this can be addressed using coarse-to-fine multi-scale implementations (see later)

- **Spatial coherency**

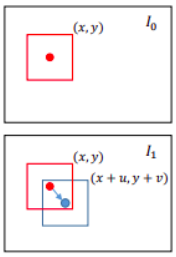
  All pixels in the template undergo the same transformation (i.e., they all lay on the same 3D surface)

  → **No errors in the template image boundaries:** only the object to track appears in the template image

  → **No occlusion**: the entire template is visible in the input image

KLT tracking applied to pure translation

Consider the reference patch centered at (x, y) in image I_0 and the shifted patch centered at $(x + u, y + v)$ in image I_1. The patch has size Ω. We want to find the motion vector (u, v) that minimizes the Sum of Squared Differences (SSD):



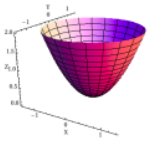
$$SSD(u, v) = \sum_{x,y \in \Omega} (I_0(x, y) - I_1(x + u, y + v))^2$$

$$I_1(x + u, y + v) \cong I_1(x, y) + I_x u + I_y v$$

$$\Rightarrow SSD(u, v) \cong \sum_{x,y \in \Omega} (I_0(x, y) - I_1(x, y) - I_x u - I_y v)^2$$

$$\Rightarrow SSD(u, v) \cong \sum_{x,y \in \Omega} (\Delta I - I_x u - I_y v)^2$$

This is a simple quadratic function in two variables (u, v)



To minimize it, we differentiate it with respect to (u, v) and equate it to zero:

$$\frac{\partial SSD}{\partial u} = 0\,, \frac{\partial SSD}{\partial v} = 0$$

$$\frac{\partial SSD}{\partial u} = 0 \Rightarrow -2 \sum I_x(\Delta I - I_x u - I_y v) = 0$$

$$\frac{\partial SSD}{\partial v} = 0 \Rightarrow -2 \sum I_y(\Delta I - I_x u - I_y v) = 0$$

$$\sum I_x(\Delta I - I_x u - I_y v) = 0$$

$$\sum I_y(\Delta I - I_x u - I_y v) = 0$$

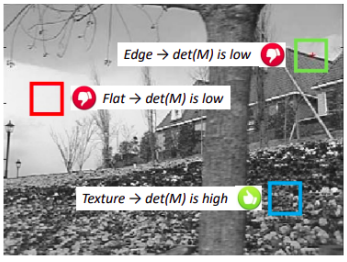
- Linear system of two equations in two unknowns
- We can write them in matrix form:

Haven't we seen this matrix already?

This is the M matrix of the Harris detector (Lecture 05)

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{bmatrix} \Rightarrow \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}^{-1} \begin{bmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{bmatrix}$$

$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$



For M to be invertible, $\det(M)$ should be non zero, which means that its eigenvalues should be large (i.e., not a flat region, not an edge) → in practice, it **should be a corner or more generally contain texture**!

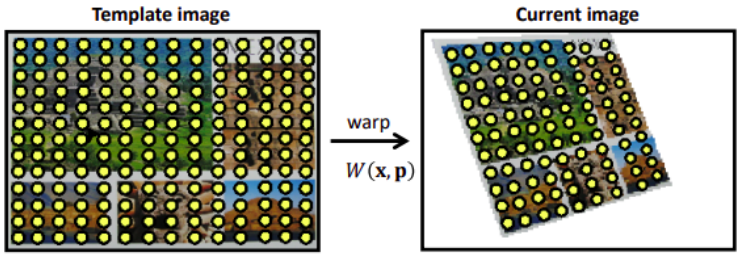
Application to Optical Flow

What if you track every single pixel in the image?

KLT applied to generic warps

Goal: estimate the parameters $\mathbf{p}$ of the transformation $W(\mathbf{x}, \mathbf{p})$ that minimize the SSD:

$$SSD = \sum_{\mathbf{x}\in\mathbf{T}} \left[I\big(W(\mathbf{x}, \mathbf{p})\big) - T(\mathbf{x}) \right]^2$$



* Every yellow dot in this image denotes a pixel

$$SSD = \sum_{\mathbf{x}\in\mathbf{T}} \left[I\big(W(\mathbf{x}, \mathbf{p})\big) - T(\mathbf{x}) \right]^2$$

- Assume that an initial estimate of $\mathbf{p}$ is known. Then, we want to find the increment $\Delta\mathbf{p}$ that minimizes

$$SSD = \sum_{\mathbf{x}\in\mathbf{T}} \left[I\big(W(\mathbf{x}, \mathbf{p} + \Delta\mathbf{p})\big) - T(\mathbf{x}) \right]^2$$

- First-order Taylor approximation of $I\big(W(\mathbf{x}, \mathbf{p} + \Delta\mathbf{p})\big)$ yelds to:

$$I\big(W(\mathbf{x}, \mathbf{p} + \Delta\mathbf{p})\big) \cong I\big(W(\mathbf{x}, \mathbf{p})\big) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p}$$

$\nabla I = [I_x, I_y]$ = Image gradient evaluated at $W(\mathbf{x}, \mathbf{p})$ — Jacobian of the warp $W(\mathbf{x}, \mathbf{p})$

- By replacing $I\big(W(\mathbf{x}, \mathbf{p} + \Delta\mathbf{p})\big)$ with its 1st order approximation, we get

$$SSD = \sum_{\mathbf{x}\in\mathbf{T}} \left[I\big(W(\mathbf{x}, \mathbf{p})\big) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2$$

- How do we minimize it?
- We differentiate SSD with respect to $\Delta\mathbf{p}$ and we equate it to zero, i.e., $\frac{\partial SSD}{\partial \Delta\mathbf{p}} = 0$

$$SSD = \sum_{\mathbf{x}\in\mathbf{T}} \left[I(W(\mathbf{x},\mathbf{p})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2$$

$$\frac{\partial SSD}{\partial \Delta\mathbf{p}} = 2 \sum_{\mathbf{x}\in\mathbf{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[I(W(\mathbf{x},\mathbf{p})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]$$

$$\frac{\partial SSD}{\partial \Delta\mathbf{p}} = 0$$

$$2 \sum_{\mathbf{x}\in\mathbf{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[I(W(\mathbf{x},\mathbf{p})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right] = 0 \Rightarrow$$

Notice that these are NOT matrix products but **pixel-wise** products!

$$\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}\in\mathbf{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[T(\mathbf{x}) - I(W(\mathbf{x},\mathbf{p})) \right] =$$

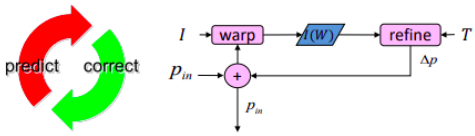
$$H = \sum_{\mathbf{x}\in\mathbf{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial W}{\partial \mathbf{p}} \right]$$

Second moment matrix (Hessian) of the warped image

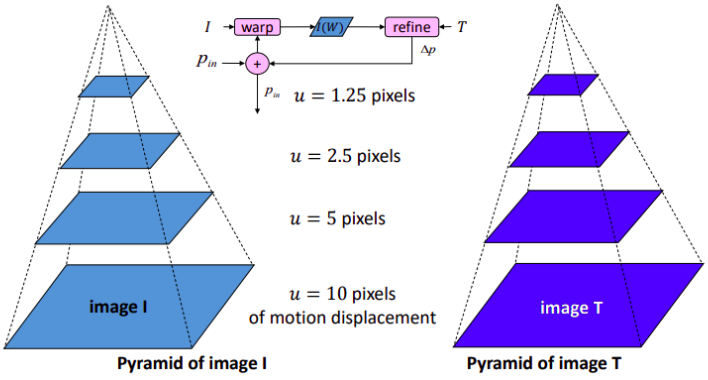
What does H look like when the warp is a pure translation?

KLT algorithm is iterated until convergence by following a **predict-correct cycle**

1. A **prediction** $I(W(\mathbf{x},\mathbf{p}))$ of the warped image is computed from an initial estimate of $\mathbf{p}$
2. The **correction** parameter $\Delta\mathbf{p}$ is then computed as a function of the **error** $T(\mathbf{x}) - I(W(\mathbf{x},\mathbf{p}))$ between the prediction and the template. The larger this error, the larger the correction applied
3. Steps 1 & 2 are iterated until the error is small and the output parameters are used as input for the next frame

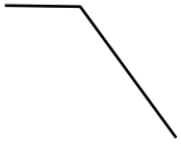


Coarse-to-fine estimation

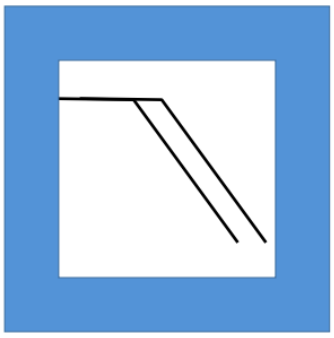


Aperture Problem

- Consider the motion of the following corner



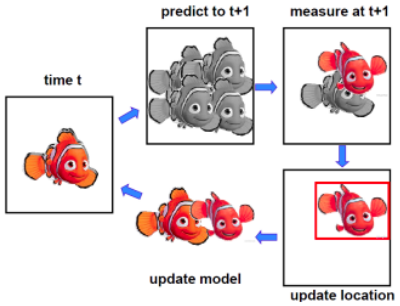
- Now, look at the local brightness changes **through a small aperture**
- We **cannot always determine** the motion direction → **Infinite motion solutions** may exist!
- Solution?
  - Increase aperture size!



- Now, look at the local brightness changes **through a small aperture**
- We **cannot always determine** the motion direction → **Infinite motion solutions** may exist!
- Solution?

Generalization of KLT

- The same concept (predict/correct) can be applied to tracking of 3D object (in this case, what is the transformation to etimate? What is the template?)
- In order to deal with wrong prediction, it can be implemented in a **Particle-Filter** fashion (using multiple hipotheses that need to be validated)



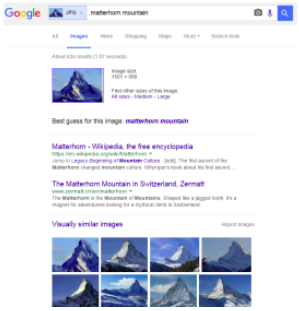
Displacement-model Jacobians ∇W_p

$$p = (a_1, a_2, \dots, a_n)$$

- Translation: $W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} x + a_1 \\ y + a_2 \end{bmatrix}$ $\quad \frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial W_1}{\partial a_1} & \frac{\partial W_1}{\partial a_2} \\ \frac{\partial W_2}{\partial a_1} & \frac{\partial W_2}{\partial a_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
- Euclidean: $W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} xcos(a_3) - ysin(a_3) + a_1 \\ xsin(a_3) + ycos(a_3) + a_2 \end{bmatrix}$ $\quad \frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} 1 & 0 & -xsin(a_3) - ycos(a_3) \\ 0 & 1 & xcos(a_3) - ysin(a_3) \end{bmatrix}$
- Affine: $W(\mathbf{x}, \mathbf{p}) = \begin{bmatrix} a_1 x + a_3 y + a_5 \\ a_2 x + a_4 y + a_6 \end{bmatrix}$ $\quad \frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$

Place Recognition

- **Robotics**:
  - Has the robot been to this place before?
  - Which images were taken around the same location?
- **Image retrieval**:
  - Have I seen this image before?
  - Which images in my database look similar to it?
    E.g., Google Reverse Image Search



Visual Place Recognition

- **Goal**: **query** an image in a database of ***N* images**
- **Complexity:** $O(NF^2)$ feature comparisons (assumes each image has ***F*** features)
  - Example:
    - assume 1,000 SIFT features per image → $F = 1{,}000$
    - assume $N = 100{,}000{,}000$
    - → $NF^2 = 100{,}000{,}000{,}000{,}000$ feature comparisons!
    - If we assume 10 microseconds per feature comparison → 1 image query would take **32 years**!

Solution: Use an index file! Complexity reduces to $O(F)$

Text Retrieval

- Image retrieval takes inspiration from **text retrieval**
- For text documents, an efficient way to find all **pages** in which a **word** occurs is to use an **index file**
- To **retrieve a given text query**, it is then sufficient to use a **voting scheme**
- Suppose that we have a **document with 10 pages** and we want to determine on which page this sequence of words appears:

"Zurich is a city of Switzerland"

- *"Zurich is a city of Switzerland"* is the **query text**
- Suppose that this is our index file:

Word	Page numbers
Zurich	3, 5, 7
City	1, 3, 5, 9
Switzerland	2, 3, 6, 8

- The solution is to use a **voting array** that has as many cells as the number of pages in the document
- We first set all the cell values to 0
- Then, we add 1 to each cell corresponding to the page numbers where the words of the query text appear according to the index file

"Switzerland" {2, 3, 6, 8}

Page number	1	2	3	4	5	6	7	8	9	10
Cell value	1	1	3	0	2	1	1	1	1	0

Bag of Words

Using the analogy from text retrieval, we need to:

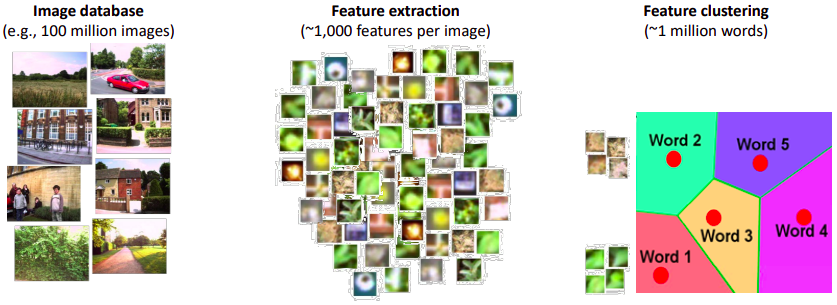
- define what a "*visual word*" is and
- define a "*vocabulary*" of visual words

This approach is known as "Bag of Words" (BOW)

- Can a SIFT descriptor be used as a visual word? And a BRISK descriptor?
  - SIFT → $128 \times 4\ bytes\ float = 512\ bytes = 4096\ bits = 2^{4096}$ possible SIFT descriptors!
  - BRISK-128 → 128 bits $= 2^{128}$ possible BRISK descriptors!
  - **Usually, 1 million visual words is enough**
  - **Idea**: **cluster SIFT descriptors into visual words**

How to extract Visual Words from descriptors

- **Collect a large enough dataset** that is representative of all possible images that are relevant to your application (e.g., for automotive place recognition, you may want to collect million of street images sampled around the world)
- Extract features and descriptors from each image and map them into the **descriptor space** (e.g., for SIFT, 128 dimensional descriptor space)
- **Cluster the descriptor space into K clusters**
- **The centroid of each cluster is a visual word**.
  - This is computed by taking the arithmetic average of all the descriptors within the same cluster:
    - e.g., for SIFT, each cluster contains SIFT features that are very similar to each other;
    - the visual word then is the average all the SIFT descriptors in that cluster

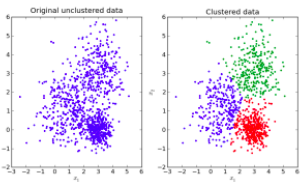


- Extracting SIFT features from a VGA images takes ~20 ms on an i7 CPU
- This means that the **extraction of features from 100 million images would take ~23 days** without accounting for the time needed for clustering all these features
- However, notice that **this is ok since this process only needs to be done once**, when the database has been created.
- **If the database grows** as new images are collected, **new features can be extracted and the visual words can be updated accordingly**. This update process **does not need to run in real time**

How do we cluster the descriptor space?

- **k-means clustering** is an algorithm to partition n data points into k clusters in which each data point $\boldsymbol{x}$ belongs to the cluster $\boldsymbol{S_i}$ with center $\boldsymbol{m_i}$
- It minimizes the **squared Euclidean distance** between points $\boldsymbol{x}$ and their nearest cluster centers $\boldsymbol{m_i}$

$$D(X, M) = \sum_{i=1}^{k} \sum_{x \in S_i} (x - m_i)^2$$



Algorithm:

- Randomly initialize k cluster centers
- Iterate until convergence:
  - Assign each data point $\boldsymbol{x_j}$ to the nearest center $\boldsymbol{m_i}$
  - Recompute each cluster center as the mean of all points assigned to it

Building the Image Vocabulary

- The **Image Vocabulary** is a data structure that lists all extracted visual words
- Each visual word is assigned a unique identifier (an **integer number**)
- **Each visual word** in the image vocabulary **points to a list of images** (from the entire image database) in which that word appears
- If the database grows, the vocabulary is updated accordingly

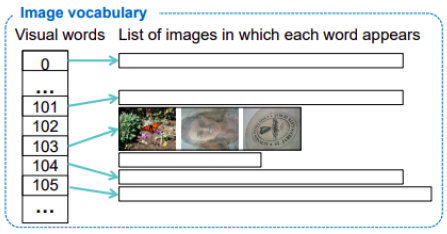


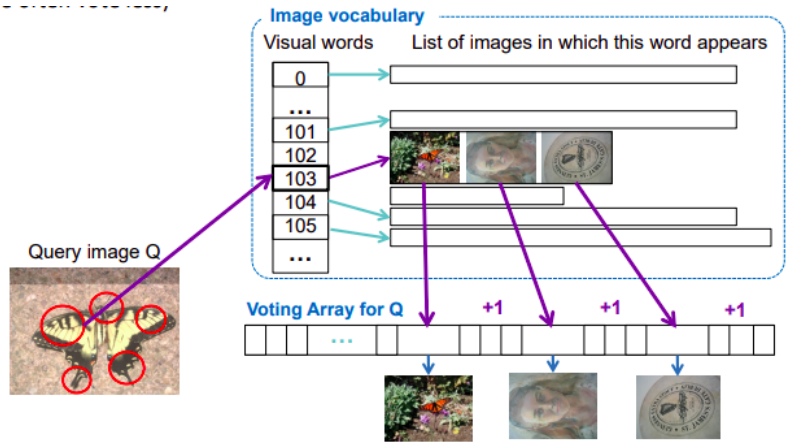
Image Retrieval

1. To query an image in the database, we must first extract features from the query image (this takes about 20 ms for ~1,000 SIFT features)

2. Then, we initialize the **voting array** to 0 (the voting array has as many cells as the number of images in the database).

3. Then, we **look-up** each feature in the image vocabulary (basically, we look for the **closest visual word** in the vocabulary)

4. Finally, **each visual word votes for multiple images** as we saw for the case of text retrieval; however, the **voting is not uniform but is weighted by the inverse of the frequency of the word** (i.e., words that repeat more often vote less)



Issues

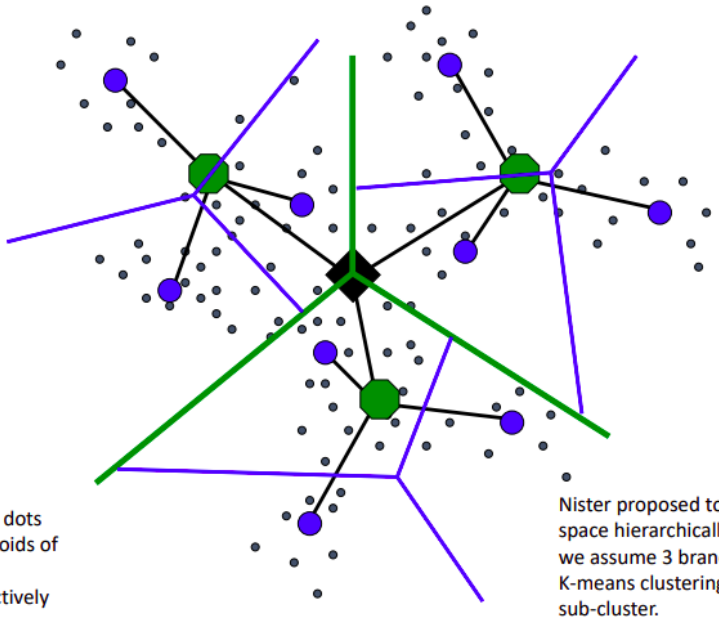
Every feature in the query image has to be compared against all visual words in the vocabulary:

- Example:
  - assume our query image has 1,000 features;
  - assume 1 million visual words → number of feature comparisons would be equal to **1 billion**!
  - If we assume 10 microseconds per feature comparison, then querying one image would take ~**3 hours**!

- How can we make the comparison cheaper?
  - Idea: use **hierarchical clustering**

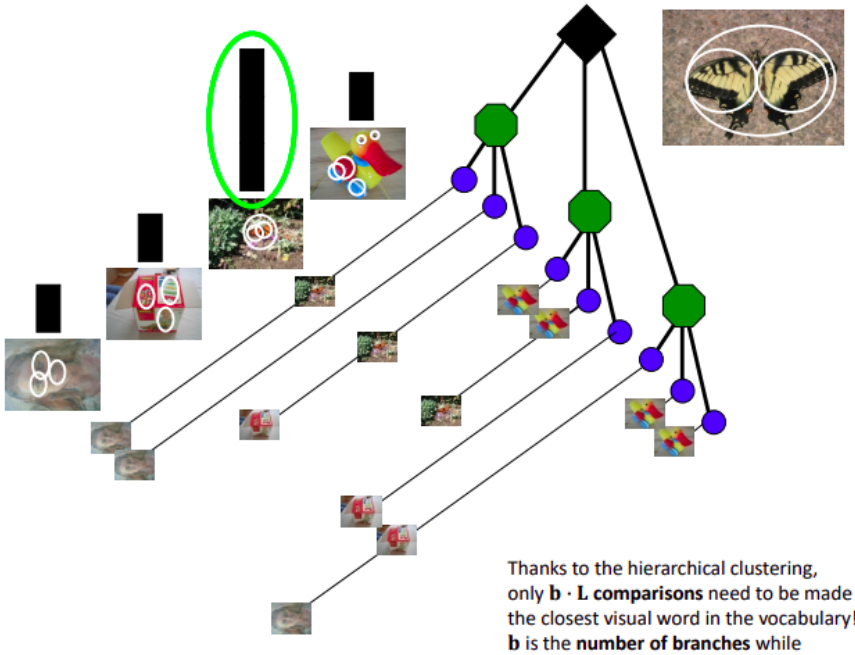
Hierarchical Clustering

- David Nister proposed to cluster the feature space in a **coarse-to-fine manner** so that visual words could be represented as the **terminal vertices of a search tree**.
- As we will see, this significantly reduces the feature-to-word association, bringing image retrieval within the reach of resource-constrained mobile devices!



The green and blue dots represent the centroids of level 1 and level 2 sub-clusters, respectively

Nister proposed to cluster the feature space hierarchically. For example, here we assume 3 branches and 2 levels. K-means clustering is used to cluster each sub-cluster.



Thanks to the hierarchical clustering, only $\mathbf{b} \cdot \mathbf{L}$ **comparisons** need to be made to find the closest visual word in the vocabulary!
b is the **number of branches** while
L is the **number of levels** of the tree.

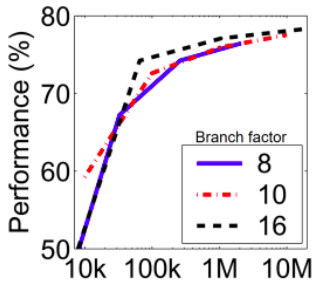
Example

Querying an image in a database of **100 million images**

- assume a query image with $\mathrm{F} = 1{,}000$ features
- assume a tree structure with $\mathrm{b} = 10$ branches per level and $\mathrm{L} = 6$ levels (i.e., $b^L = 1{,}000{,}000$ visual words)
- Then, the **number of feature comparisons** $= \mathbf{F} \cdot \mathbf{b} \cdot \mathbf{L} = 1{,}000 \cdot 10 \cdot 6 = 60{,}000$ **instead of** $\mathbf{F} \cdot \boldsymbol{b}^{\boldsymbol{L}} =$ 1 billion comparisons (which was the case without hierarchical clustering)
- If we assume 10 microseconds per feature comparison
  → 1 image query would take **0.6 seconds**!

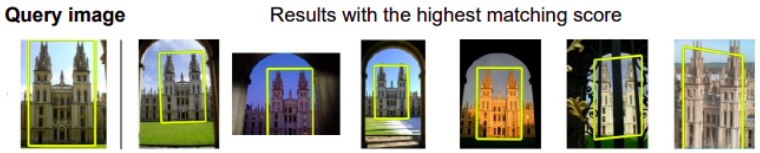
How many visual words, branches, and levels?

- **More words is better**, but **1 million words are used in practice**
- Also, **6 branches and 10 depth levels are practically used** (e.g., ORB-SLAM)



Geometric Verification

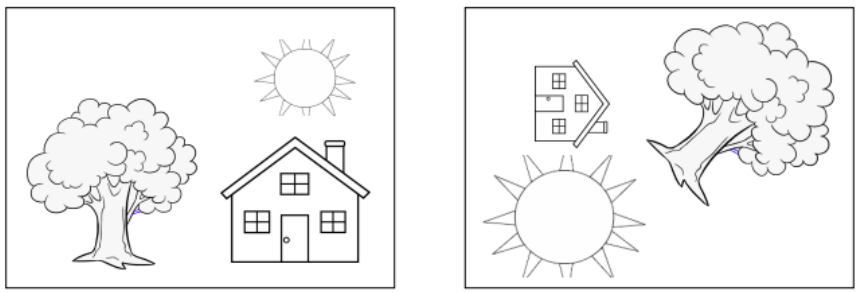
Bag of Words returns a list of images with score above a given threshold. How do we pick the image that was captured **closest** to the query image?



- Observe that the visual vocabulary discards the spatial relationships between features
- Solution: Test each returned image for geometric consistency against the query image (e.g. using 5- or 8-point RANSAC) and pick the image with **the smallest reprojection error and largest number of inliers**

Curiosity

Imagine to query two images with the same features shuffled around. Will the scores returned by Bag of Words be different?



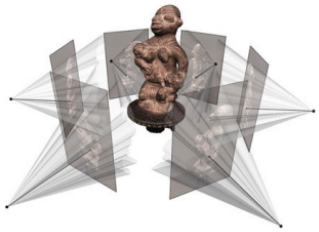
Open Challenges

When does place recognition fail and how would you address them?

Dense Reconstruction (or Multi-view stereo)

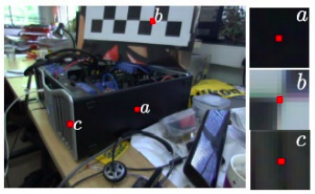
Problem definition:

- **Input**: calibrated images from several viewpoints (i.e., K, R, T are known for each camera, e.g., from SFM)
- **Output**: 3D object **dense** reconstruction (ideally of every pixel)



Challenges

- **Dense reconstruction** requires establishing **dense correspondences**
- But **not all pixels can be matched** reliably:
  - flat regions,
  - edges,
  - viewpoint and illumination changes,
  - occlusions



Idea: Take advantage of many small-baseline views where high-quality matching

Dense reconstruction workflow

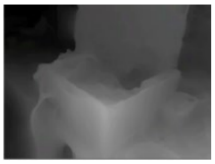
Step 1: Local methods →

- Estimate depth independently for each pixel

How do we compute correspondences for *every* pixel?

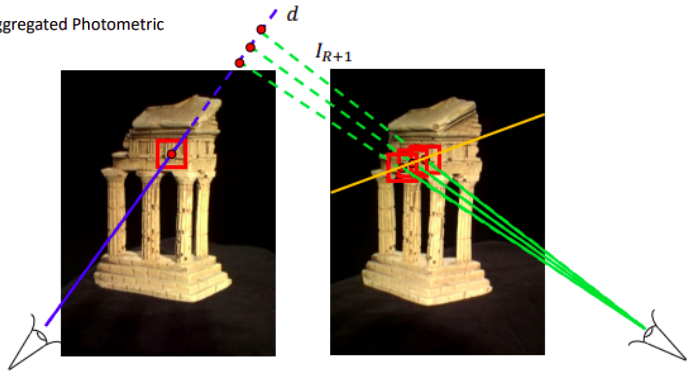
Step 2: Global methods →

- Refine the *depth map as a* whole by enforcing smoothness. This process is called *regularization*

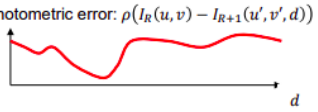


Solution: Aggregated Photometric Error

Set the first image as reference and estimate depth at each pixel by minimizing the Aggregated Photometric Error in all subsequent frames



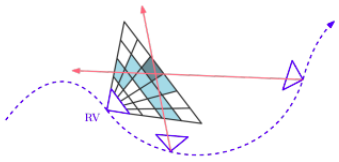
Photometric error: $\rho(I_R(u,v) - I_{R+1}(u',v',d))$



This error term is computed for between the reference image and each subsequent frame. The sum of these error terms is called Aggregated Photometric Error (see next slide)

Disparity Space Image (DSI)

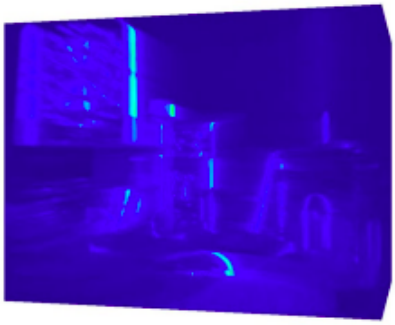
- Image resolution: 240×180 pixels
- Number of disparity (depth) levels: 100
- DSI:
  - size: $240 \times 180 \times 100$ voxels; each voxel contains the Aggregated Photometric Error $C(u,v,d)$ (see next slide)
  - white = high Aggregated Photometric Error
  - blue = low Aggregated Photometric Error



Non-uniform, projective grid, centered on the reference frame I_R

Reference image

DSI (dark means high)



Disparity Space Image (DSI)

- For a given image point (u, v) and for discrete depth hypotheses d, the **Aggregated Photometric Error** $C(u, v, d)$ with respect to the reference image I_R can be stored in a volumetric 3D grid called the **Disparity Space Image (DSI)**, where each voxel has value:

$$C(u, v, d) = \sum_{k=R+1}^{R+n-1} \rho\big(I_R(u, v) - I_k(u', v', d)\big)$$

where n is the number of images considered and $I_k(u', v', d)$ is the patch of intensity values in the k-th image centered on the pixel (u', v') corresponding to the patch $I_R(u, v)$ in the reference image I_R and depth hypothesis d; thus, formally:

$$I_k(u', v', d) = I_k\left(\pi\left(T_{k,R}(\pi^{-1}(u, v) \cdot d)\right)\right)$$

where $T_{k,R}$ is the relative pose between frames R and K

- $\rho(\cdot)$ is the photometric error (SSD) (e.g. L_1, L_2, Tukey, or Huber norm)

Depth estimation

The solution to the depth estimation problem is to find a **function $d(u, v)$** (called **depth map**) in the DSI that minimizes the **aggregated photometric error:**

$$depth\ map = d(u, v) = \arg\min_d \sum_{(u,v)} C(u, v, d(u, v))$$

How does this relate to stereo vision?

Depth map: each pixel intensity encodes the depth of that pixel. Dark = far, bright = close.

- Smaller window
  - More detail
  - More noise
- Larger window
  - Smoother disparity maps
  - Less detail

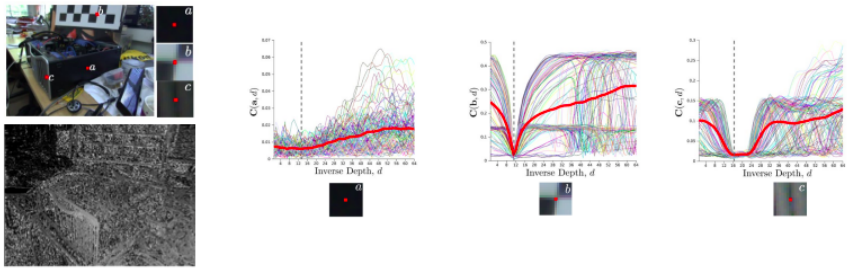
Influence of patch size

$W = 3$ $W = 20$

Can we use a patch size of 1×1 pixels?

Influence of the patch appearance

- Aggregated photometric error for **flat regions** (a) and **edges parallel to epipolar lines** (c) **show flat valleys (plus noise)**
- For **textured areas** (e.g., corners (b) or blobs), the aggregated photometric error has one **distinctive minimum**
- **Repetitive texture** shows **multiple minima**



Regularization

To penalize wrong reconstruction due to image noise and ambiguous texture, we add a smoothing term (called regularization term) to the optimization:

$$d(u, v) = \arg\min_d \textstyle\sum_{(u,v)} C(u, v, d(u, v))) \quad \text{(local methods)}$$

subject to

Piecewise smooth (global methods)

Regularization

- Formulated in terms of energy minimization
- The objective is to find a **surface** $d(u, v)$ that minimizes a global energy functional:

$$E(d) = \underbrace{E_d(d)}_{\text{Data term}} + \underbrace{\lambda E_s(d)}_{\text{Regularization term (i.e., smoothing)}}$$

where λ controls the tradeoff between data and regularization. What happens as λ increases?

Data term: $E_d(d) = \sum_{(u,v)} C(u, v, d(u, v))$

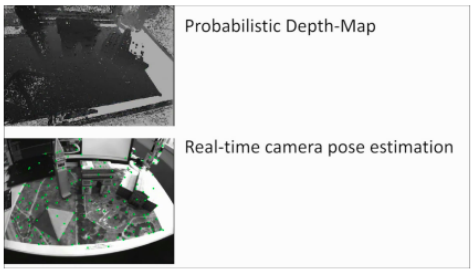
Regularization term: $E_s(d) = \sum_{(u,v)} \left(\frac{\partial d}{\partial u}\right)^2 + \left(\frac{\partial d}{\partial v}\right)^2$

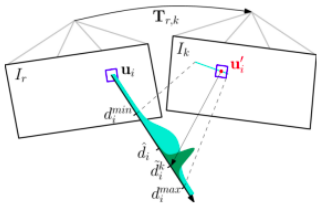
How can we deal with depth discontinuities between separate objects?

The regularization term $E_s(d)$ basically fills the holes: it **smooths the depth map** by softing discontinuities

Probabilistic Monocular Dense Reconstruction

- Estimate depth uncertainty
- **Regularize only 3D points** with **low depth uncertainty** (does not fill holes if present, which is good for robotic applications)



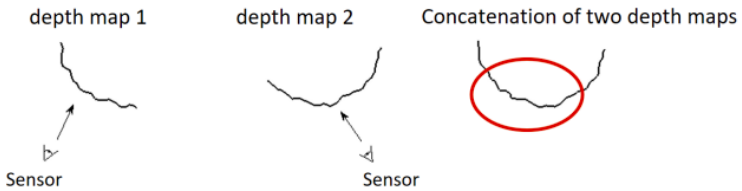


Pizzoli, Forster, Scaramuzza, *REMODE: Probabilistic, Monocular Dense Reconstruction in Real Time*, IEEE International Conference on Robotics and Automation (ICRA), 2014. PDF. Video. Code.

23

What about large motions?

- When the **distance between the current frame and the reference frame** gets too large (e.g., $> 10\%$ of the average scene depth), then the Aggregated Photometric Error starts to diverge due to the **large view point changes**
- Solution: **create a new reference frame** (keyframe) and start a new depth map computation



GPU for 3D Dense Reconstruction

- **Image processing**
  - Filtering & Feature extraction (i.e., **convolutions**)
  - Warping (e.g., **epipolar rectification**, **homography**)
- **Multiple-view geometry**
  - Search for **dense correspondences**
    - Pixel-wise operations (SAD, **SSD**, NCC)
    - **Matrix and vector operations** (epipolar geometry)
  - **Aggregated Photometric Error** for multi-view stereo
- **Global optimization**
  - Variational methods (i.e., **regularization** (smoothing))
    - **Parallel, in-place** operations for gradient / divergence computation
- **Deep learning**

GPU: Graphics Processing Unit

optimized for **serial processing** (i.e., only a few instructions can be executed during the same clock

forms calculations **in parallel** on thousands of cores (i.e., thousands of instructions can be executed ıe same clock cycle)

ALU: Arithmetic Logic Unit

Supervised Learning

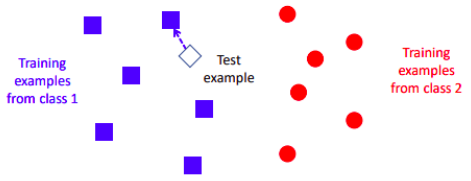
Find function $f(x, \theta)$ that imitates a ground truth signal

Machine Learning Keywords

1. **Loss**: Quantify how good θ are
2. **Optimization**: The process of finding θ that minimize the loss
3. **Function**: Problem modelling → Deep networks are highly non-linear $f(x, \theta)$

Classifiers: K-Nearest neighbor

Features are represented in the descriptor space



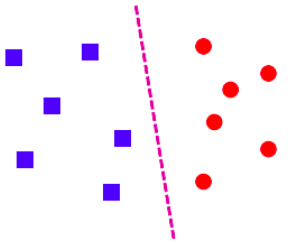
$f(\boldsymbol{x}, \theta)$ = label of the K training examples nearest to $\boldsymbol{x}$

How fast is training? How fast is testing?
- O(1), O(n)

What is a good distance metric ? What K should be used? ☹

Classifiers: Linear

Find a *linear function* to separate the classes:



$$f(\boldsymbol{x}, \theta) = sgn(\theta \cdot \boldsymbol{x} + b)$$

What is θ? What is the dimensionality of images?

Multi layer perceptron (non linear classifier)
- Composition of linear layer with non linear activation functions
- **Back-propagation**: changes weights using gradients of loss with respect to weights (stochastic gradient descent)
- Too many parameters → possible **overfitting**

Convolutional Neural Networks
- Use masks (shared weights) for spatial consistency
- Going deep improve performances → extract progressively more sophisticated information

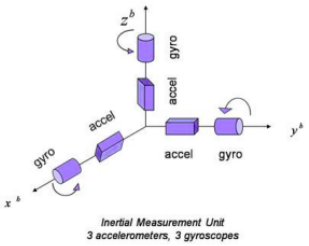
Supervised Learning
- We have access to both input data or images and ground truth labels
- Network trained with supervision ususally perform the best
- Data generation and hand labeling is hard
- Usual tasks are image segmentation and image captioning

Unsupervised Learning
- only access to input data or images → can use larger datasets (no manual labeling needed)
- examples are monocular depth estimation, structure from motion

	Supervised	**Unsupervised**
Performance	Usually better for the same dataset size.	Usually worse, but can outperform supervised methods due to larger data availability.
Data availability	Low, due to manual labelling.	High, no labelling required.
Training	Simple, ground truth gives a strong supervision signal.	Sometimes difficult, loss functions have to be engineered to get good results.
Generalizability	Good, although sometimes the network learns to blindly copy the labels provided, leading to poor generalizability.	Better, since unsupervised losses often encode the task in a more fundamental way.

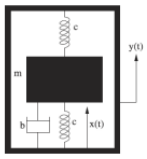
What is an IMU?

- **I**nertial **M**easurement **U**nit
  - Gyroscope: Angular velocity
  - Accelerometer: Linear Accelerations



Inertial Measurement Unit
3 accelerometers, 3 gyroscopes

Mechanical Gyroscope

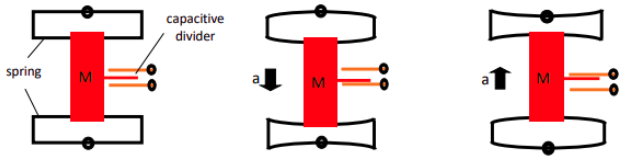


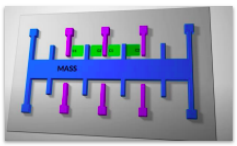
Mechanical Accelerometer

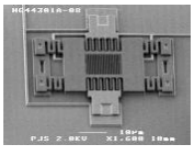
- Different categories
  - Mechanical ($100,000-1M)
  - Optical ($20,000-100k)
  - MEMS (from 1$ (phones) to 1,000$ (higher cost because they have a microchip running a Kalman filter))
- For small mobile robots & drones: MEMS IMU are mostly used
  - Cheap
  - Power efficient
  - Light weight and solid state

MEMS Accelerometer

A spring-like structure connects the device to a seismic mass vibrating in a capacitive divider. A capacitive divider converts the displacement of the seismic mass into an electric signal. Damping is created by the gas sealed in the device.







Why do we need an IMU?

- Monocular vision is **scale ambiguous**
- Pure vision is **not robust enough**
  - Low texture
  - High Dynamic Range (HDR)
  - High speed motion

Robustness is a critical issue: Tesla accident

"The autopilot sensors on the Model S failed to distinguish a white tractor-trailer crossing the highway against a bright sky." [The Guardian]

Motion blur

High Dynamic Range

Why is an IMU alone not enough?

- **Pure IMU integration will lead to large drift** (especially in cheap IMUs)
- Example: **1D scenario**. Double integration of acceleration returns the position:

$$x(t) = x_0 + v_0(t - t_0) + \iint_{t_0}^{t} a(\tau) d\tau^2$$

- If there is a **constant bias in the acceleration**, the error of position will be **proportional to t^2**
- Similarly for the orientation: if there is a **bias in angular velocity**, the error is **proportional to the time t**

GRADE/TIME	1 s	10 s	60 s	10 min	1 hr
Consumer	6 cm	6.5 m	400 m	200 km	39,000 km
Industrial	6 mm	0.7 m	40 m	20 km	3,900 km
Tactical	1 mm	8 cm	5 m	2 km	400 km
Navigation	<1 mm	1 mm	50 cm	100 m	10 km

Automotive, smartphones, and drones accelerometers

Table from Vectornav, one of the best IMU companies. Errors were computed assuming the device at rest:
https://www.vectornav.com/resources/inertial-navigation-primer/specifications--and--error-budgets/specs-inserrorbudget

Why visual inertial fusion?

IMU and vision are complementary

Cameras

- **Exteroceptive sensor**: measures light energy from the environment
- × Sensitive to motion blur, HDR, texture
- ✓ Drift is bounded when observing the same scene
- ✓ Precise in slow motion
- × Limited output rate (~100 Hz)
- × Scale ambiguity in monocular setup

IMU

- **Proprioceptive sensor**: measures values internal to the system
- ✓ Insensitive to motion blur, HDR, texture
- × Drift grows unbounded regardless of the environment
- × Less precise in slow motion (low signal-to-noise ratio)
- ✓ High output rate (1,000-10,000 Hz)

- ✓ Can be used as a prior to predict next feature positions

What cameras and IMU have in common: both can be used to estimate the pose incrementally (known as dead-reckoning), which suffers from drift over time. **Solution: fuse them together to reduce drift (see later)**

IMU Measurement Model

The model measures angular velocity $\tilde{\omega}_B(t)$ and acceleration $\tilde{a}_B(t)$ in the body frame B:

$$\begin{aligned} \tilde{\omega}_B(t) &= \omega_B(t) + b^g(t) + n^g(t) \\ \tilde{a}_B(t) &= R_{BW}(t)(a_W(t) - g) + b^a(t) + n^a(t) \end{aligned}$$

IMU biases + noise in body frame

Raw IMU measurements

true $\boldsymbol{\omega}$ (in body frame) and true **a** (in world frame) to estimate

Notation:

- The superscript g stands for gyroscope and a for accelerometer
- R_{BW} is the rotation of the World frame W with respect to Body frame B
- The gravity g is expressed in the World frame
- Biases and noise are expressed in the body frame

- Additive, zero-mean Gaussian white noise: $n^g(t)$, $n^a(t)$
- Biases: $b^g(t)$, $b^a(t)$
  - The gyroscope and accelerometer biases are considered **slowly varying "constants"**. Their temporal fluctuation is modeled by saying that the derivative of the bias is a zero-mean Gaussian noise with standard deviation σ_b

$$\dot{\mathbf{b}}(t) = \sigma_b \mathbf{w}(t) \qquad \mathbf{w}(t) \sim \mathbf{N}(0,1)$$

- Some facts about IMU biases:
- They change with **temperature** and **mechanical and atmospheric pressure**
- Thus they **may also be different every time the IMU is turned on**
- Good news: **they can be estimated!** (see later)

IMU Integration Model

- The **IMU Integration Model** computes the position, orientation, and velocity of the IMU in the world frame. To do this, we must first compute the acceleration $a(t)$ in the world frame from the measured one $\tilde{a}(t)$ in the body frame (see Slide 13):

$$a(t) = R_{WB}(t)\big(\tilde{a}(t) - b(t)\big) + g$$

- The position p_k at time t_k can then be **predicted** from the position p_{k-1} at time t_{k-1} by integrating all the inertial measurements $\{\tilde{a}_j, \tilde{\omega}_j\}$ within that time interval:

$$p_k = p_{k-1} + v_{k-1}(t_k - t_{k-1}) + \iint_{t_{k-1}}^{t_k} \big(R_{WB}(t)\big(\tilde{a}(t) - b(t)\big) + g\big)dt^2$$

- A similar expression can be obtained to predict the velocity v_k and orientation R_{WB} of the IMU in the world frame as functions of both $\tilde{a}_j$ and $\tilde{\omega}_j$

For convenience, the **IMU Integration Model** is normally written as

$$\begin{pmatrix} p_k \\ q_k \\ v_k \end{pmatrix} = f\begin{pmatrix} p_{k-1} \\ q_{k-1}, u \\ v_{k-1} \end{pmatrix} \quad \text{or, more compactly:} \quad x_k = f(x_{k-1}, u)$$

where:

- $x = \begin{bmatrix} p \\ q \\ v \end{bmatrix}$ represents the IMU **state**, i.e., **position, orientation, and velocity**
- q is the IMU **orientation $\boldsymbol{R_{WB}}$** (usually represented using quaternions)
- $u = \{\tilde{a}_j, \tilde{\omega}_j\}$ are the accelerometer and gyroscope measurements in the time interval $[t_{k-1}, t_k]$

Visual Inertial Odometry

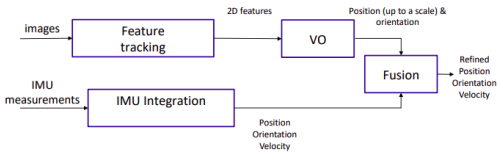
L are the landmarks (3D points)

Different paradigms exist:

- **Loosely coupled**:
  - Treats **VO and IMU as two separate black boxes** (not coupled)
    - Each black box estimates pose and velocity from visual (up to a scale) and inertial data (absolute scale)
    - Easy to implement
    - Inaccurate. Should not be used
- **Tightly coupled**:
  - Makes use of the **raw sensors' measurements** (2D features and IMU readings)
    - More accurate
    - More implementation effort

In this lecture, we will only see **tightly coupled approaches**

The Loosely Coupled Approach

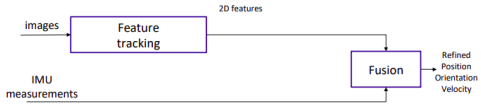


- System states:

Tightly coupled: $\mathrm{X} = [p(t);\ q(t);\ v(t);\ \mathrm{b}^a(t);\ \mathrm{b}^g(t);\ L_1;\ L_2;\ \dots;\ L_k]$

Loosely coupled: $\mathrm{X} = [p(t);\ q(t);\ v(t);\ \mathrm{b}^a(t);\ \mathrm{b}^g(t)]$

The Tightly Coupled Approach

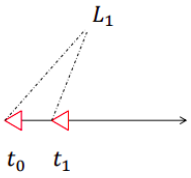


Closed-form Solution (1D case)

- From a **single camera** we only get the absolute position x up to a **unknown scale factor** s, thus

$$x = s\tilde{x}$$

where $\tilde{x}$ is the relative motion estimated by the camera.



- From the IMU

$$x = x_0 + v_0(t_1 - t_0) + \iint_{t_0}^{t_1} a(t)dt^2$$

- By equating them

$$s\tilde{x} = x_0 + v_0(t_1 - t_0) + \iint_{t_0}^{t_1} a(t)dt^2$$

- As shown in [Martinelli'14], if we assume to know x_0 (usually we set it to 0), then, for 6DOF, **both s and v_0 can be determined in closed form** from a **single feature observation and 3 views**

Non-linear Optimization Methods

VIO is solved as a non-linear Least Square optimization problem over:

$$\{X, L, b^a, b^g\} = argmin_{\{X, L, b^a, b^g\}} \left\{ \sum_{k=1}^{N} \|f(x_{k-1}, u) - x_k\|^2_{\Lambda_k} + \sum_{k=1}^{N}\sum_{i=1}^{M} \|\pi(x_k, L_i) - z_{i_k}\|^2_{\Sigma_{i_k}} \right\}$$

IMU residuals — *Reprojection residuals* (Bundle Adjustment term)

where

- $X = \{x_1, \dots x_N\}$: set of **state estimates** x_k **(position, velocity, orientation)** at frame times k
- $L = \{L_1, \dots, L_M\}$: **3D landmarks**
- $f(x_{k-1}, u)$: **state prediction** obtained by integrating IMU measurements $u = \{\tilde{a}_j, \tilde{\omega}_j\}$
- $\pi(x_k, l_i)$: **expected measurement at state estimates** x_k from projection of landmark L_i onto camera frame I_k
- z_{i_k}: **observed features**
- Λ_k: **state covariance** from the IMU integration $f(x_{k-1}, u)$
- Σ_{i_k}: is the covariance from the noisy 2D feature measurements

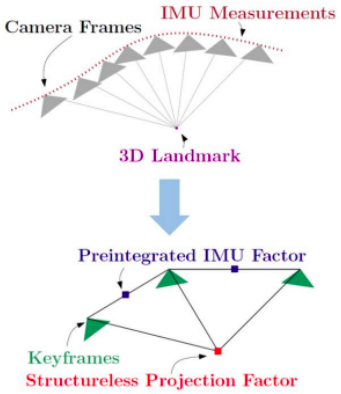
Case Study 1: OKVIS

Because the complexity of the optimization is cubic with respect to the number of cameras poses and features, real-time operation becomes infeasible as the trajectory and the map grow over time, OKVIS proposed to only **optimize the current pose and a window of past keyframes**

Case Study 2: SVO+GTSAM

Solves the same optimization problem as OKVIS but:

- **Keeps all the frames** (from the start to the end of the trajectory)
- To make the optimization efficient
  - **Marginalizes 3D landmarks** (rather than triangulating landmarks, only their visual bearing measurements are kept and the visual constraint are enforced by the **Epipolar Line Distance** (Lecture 08))
  - **pre-integrates the IMU** data between keyframes (see later)
- **Optimization solved using Factor Graphs** via GTSAM
  - Very fast because it only **optimizes** the **poses** that are **affected by a new observation**



Problem with IMU integration

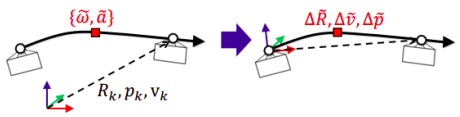
- The integration of IMU measurements, $f(x_{k-1}, u)$, from $k-1$ to k is related to the state esti time $k-1$
- During optimization, every time the linearization point at $k-1$ changes, the integration betv and k must be re-evaluated, thus slowing down the optimization

$$\{X, L, b^a, b^g\} = argmin_{\{X, L, b^a, b^g\}} \left\{ \sum_{k=1}^{N} \|f(x_{k-1}, u) - x_k\|^2_{\Lambda_k} + \sum_{k=1}^{N} \sum_{i=1}^{M} \|\pi(x_k, L_i) - z_{i_k}$$

IMU residuals — *Reprojection residuals* (Bundle Adjustment term

- **Idea: Preintegration**
  - defines relative motion increments, expressed in body frame, which are independent on position, orientation, and velocity at k [1]
  - [2] uses this theory by leveraging the manifold structure of the rotation group SO(3)

IMU Pre-Integration



Standard:
Evaluate **error in global frame**:

$e_R = \hat{R}(\tilde{\omega}, R_{k-1})^T R_k$

$e_v = \hat{v}(\tilde{\omega}, \tilde{a}, v_{k-1}) - v_k$

$e_p = \underbrace{\hat{p}(\tilde{\omega}, \tilde{a}, p_{k-1})}_{\text{Prediction}} - \underbrace{p_k}_{\text{Estimate}}$

Repeats integration when previous state changes!

Preintegration:
Evaluate **relative errors (i.e., in body frame)**:

$e_R = \Delta\tilde{R}^T \Delta R$

$e_v = \Delta\tilde{v} - \Delta v$

$e_p = \Delta\tilde{p} - \Delta p$

Preintegration of IMU deltas possible with **no initial condition required**.

Non-linear Optimization methods	Filtering methods
Optimize a **window** of multiple states (or all the states) using non-linear Least-Squares optimization	Solve the same problem by **running only one iteration of the optimization** function (e.g., using Extended Kalman Filter (EKF))
✓Multiple iterations (it re-linearizes at each iteration)	× One iteration only
✓Achieves the highest accuracy	× Sensitive to linearization point
✓Slower	✓Fastest

Case Study 1: ROVIO

- EKF state: $\mathrm{X} = [p(t);\ q(t);\ v(t);\ \mathrm{b}^a(t);\ \mathrm{b}^g(t);\ L_1;\ L_2; \dots;\ L_k]$
- Basic idea:
  1. **Prediction step**: predicts next position, velocity, orientation, and features using IMU integration model
  2. **Measurement update**: refines state by leveraging visual constraint (ROVIO minimizes the photometric error between corresponding points (alternative would be the reprojection error))
- **Complexity** of the EKF grows **quadratically** in the number of estimated landmarks
  - Thus, **max 20 landmarks** are tracked to allow real-time operation
- **Only updates the most recent state**

Case Study 2: MSCKF

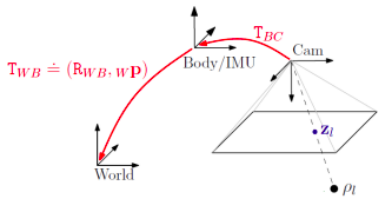
- **MSCKF (Multi-State Constraint Kalman Filter)** updates **multiple past poses** $\{p_{\mathrm{C}_1}, q_{\mathrm{C}_1}, \dots, p_{\mathrm{C}_N}, q_{\mathrm{C}_N}\}$ in addition to the current state $\{p(t), q(t), v(t)\}$. State vector:

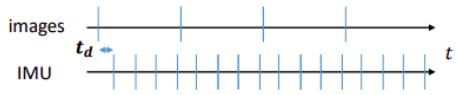
$$\mathrm{X} = \left[p(t);\ q(t);\ v(t);\ \mathrm{b}^a(t);\ \mathrm{b}^g(t);\ p_{\mathrm{C}_1};\ q_{\mathrm{C}_1};\ \dots;\ p_{\mathrm{C}_N};\ q_{\mathrm{C}_N}\right]$$

- **Prediction step:** same as ROVIO
- **Measurement update:**
  - Differently from ROVIO,
    - **landmark positions** are **not added to the state vector**, thus can run very fast independently of the number of features
    - Visual constraint is obtained from the **Epipolar Line Distance** (Lecture 08)
- Used in spacecraft landing (**NASA/JPL Moon and Mars landing**), **DJI drones**, **Google ARCore**
- Released open source within the OpenVins project: https://github.com/rpng/open_vins

Camera-IMU Calibration

- **Goal**: estimate the **rigid-body transformation** $\boldsymbol{T_{BC}}$ and time **offset** $\boldsymbol{t_d}$ **between** the **camera** and the **IMU** caused by communication delays and the internal sensor delays (introduced by filters and logic).
- **Assumptions**: Camera and IMU rigidly attached. Camera intrinsically calibrated.
- **Data**:
  - Image points from calibration pattern (checkerboard or QR board)
  - IMU measurements: accelerometer $\{a_k\}$ and gyroscope $\{\omega_k\}$





Kalibr Toolbox

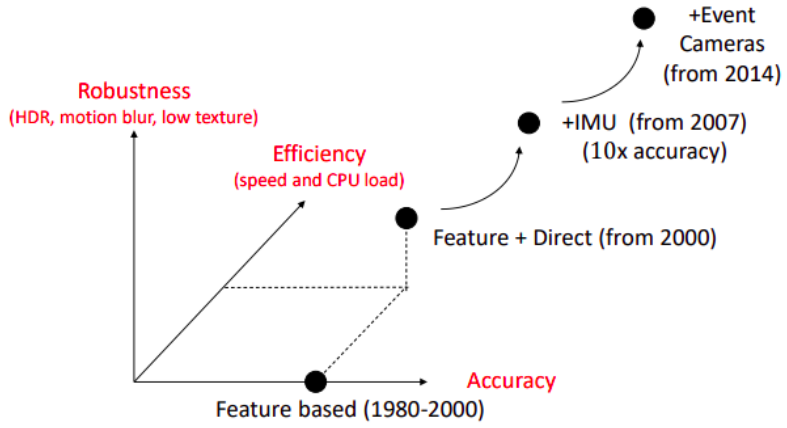
- Solves a non-linear Least Square optimization problem similar to that seen before but also optimizes over T_{BC}, t_d:

$$\{X, L, T_{BC}, t_d, b^a, b^g\} = argmin_{\{X, L, T_{BC}, t_d, b^a, b^g\}} \left\{ \sum_{k=1}^{N} \|f(x_{k-1}, u) - x_k\|^2_{\Lambda_k} + \sum_{k=1}^{N} \sum_{i=1}^{M} \|\pi(x_k, L_i) - z_{i_k}\|^2_{\Sigma_{i_k}} \right\}$$

IMU residuals — *Reprojection residuals* (Bundle Adjustment term)

- Continuous-time modelling using splines for X
- Numerical solver: Levenberg-Marquardt

A Short Recap of the last 30 years of VIO



Challenges are:
- latency and motion blur
- dynamic range

Event cameras do not suffer from these problems.

They are inspired by human eyes that have many photoreceptors but relatively less axioms.

They offer higher sensitivity (for example in low light conditions)

What is an Event Camera

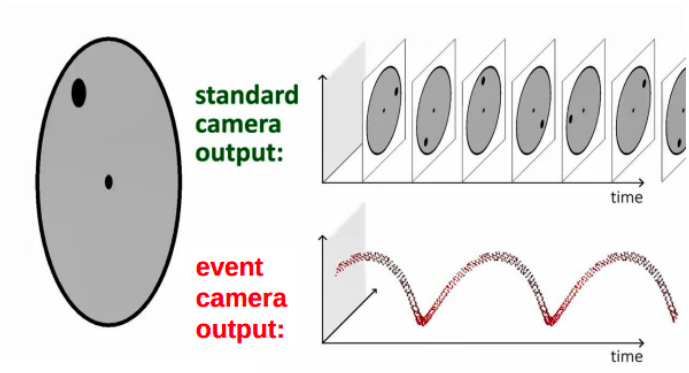
First commercialized by Prof. T. Delbruck in 2008 at the Institute of Neuroinformatics of UZH & ETH under the name of Dynamic Vision Sensor (DVS)

Advantages

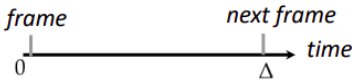
- **Low-latency** (~1 micro-seconds)
- **High-dynamic range (HDR)** (140 dB instead 60 dB)
- **High updated rate** (1 MHz)
- **Low power** (10mW instead 1W)

Challenges

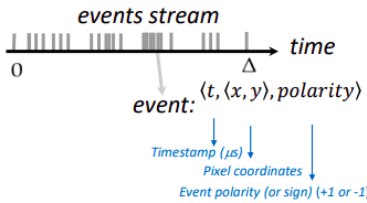
- **Paradigm shift**: Requires totally **new vision algorithms** because:
  - **Asynchronous** pixels
  - **No intensity information** (only binary intensity changes)



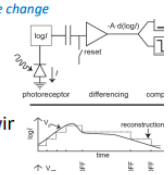
- A **traditional camera** outputs frames at **fixed time intervals**:



- By contrast, an event camera outputs **asynchronous events** at **microsecond resolution**. An event is generated each time a single pixel detects a change of intensity

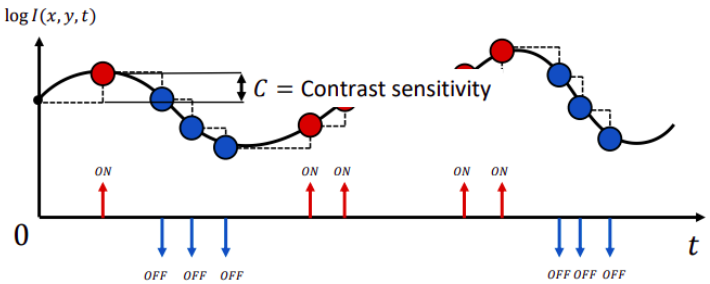


Generative Event Model



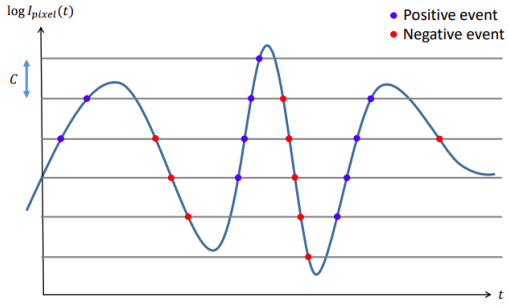
- Consider the intensity at a **single pixel** (x, y). An event is generated when the followir satisfied:

$$\log I(x, y, t + \Delta t) - \log I(x, y, t) = \pm C$$

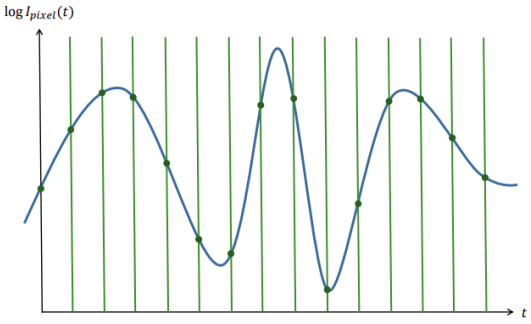


Events are triggered **asynchronously**

Event cameras sample intensity when this crosses a threshold
(Level-crossing sampling)



Standard cameras sample intensity at uniform time intervals
(uniform time sampling)



Current commercial applications

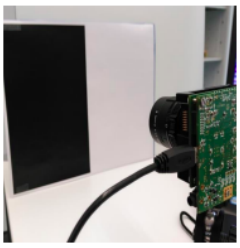
- **Monitoring and surveillance**
  - Action and gesture recognition in HDR scenes
- **Industrial automation**
  - Fast object counting
- **Computational photography**
  - Deblurring, super resolution, HDR, slow-motion video
- **Automotive**:
  - low-latency detection, object classification, low-power and low-memory storage

Calibration of an Event Camera

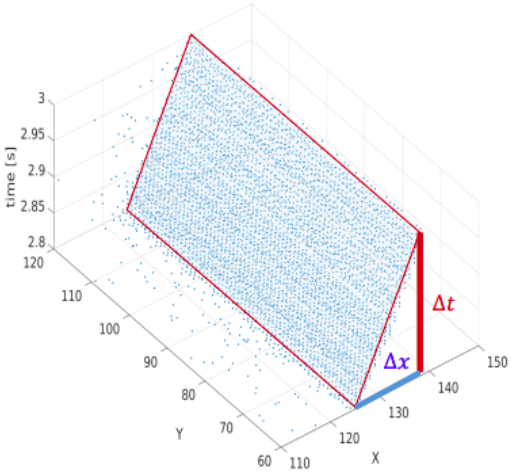
- Standard **pinhole camera model** still valid (same optics)
- Standard passive calibration patterns **cannot be used**
  - need to move the camera ⟶ inaccurate corner detection
- **Blinking patterns** (computer screen, LEDs)
- ROS DVS driver + intrinsic and extrinsic mono & stereo calibration: https://github.com/uzh-rpg/rpg_dvs_ros

A Simple Optical Flow Algorithm

- Let's assume pure horizontal motion
- White pixels become black ⟶ brightness decrease ⟶ negative events (in black color)



- The same edge, visualized in space-time
- Events are represented by dots



The edge is moving at a speed of:

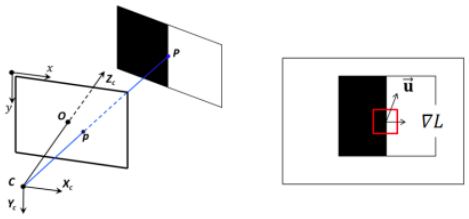
$$v = \frac{\Delta x}{\Delta t}$$

1st order approximation of the Generative Event Model

- An event is generated when the following condition is satisfied:

$$\log I(x, y, t + \Delta t) - \log I(x, y, t) = \pm C$$

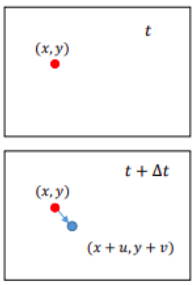
- For many applications, it is convenient to derive a 1st order approximation
- Let us define $L(x, y, t) = Log(I(x, y, t))$
- Consider a given pixel $p(x, y)$ with gradient $\nabla L(x, y)$ undergoing the motion $\boldsymbol{u} = (u, v)$ in pixels, induced by a moving 3D point $\boldsymbol{P}$



25

- Let's apply the **brightness constancy assumption**, which says that the intensity value of p before and after the motion must remain unchanged:

$$L(x, y, t) = L(x + u, y + v, t + \Delta t)$$



- By replacing the right-hand term with its 1st order approximation at $t + \Delta t$, we get:

$$L(x, y, t) = L(x, y, t + \Delta t) + \frac{\partial L}{\partial x} u + \frac{\partial L}{\partial y} v$$

$$\Rightarrow L(x, y, t + \Delta t) - L(x, y, t) = -\frac{\partial L}{\partial x} u - \frac{\partial L}{\partial y} v$$

$$\Rightarrow \boxed{\pm C = -\nabla L \cdot \boldsymbol{u}}$$

- This formula shows that **maximum generation of events** (i.e., higher event rate) occurs when the **relative motion of the camera is perpendicular to the edge** and is **minimum when parallel** to the edge.

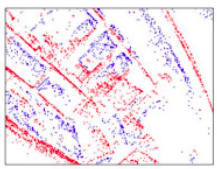
Application 1: Image Reconstruction from events

- Probabilistic **simultaneous gradient reconstruction and rotation estimation** from $\pm C = -\nabla L \cdot \boldsymbol{u}$
- Obtain **image intensity from gradient** via Poisson reconstruction
- The reconstructed image has **super-resolution and High Dynamic Range** (HDR)
- Can run in **real time on a GPU**

Application 2: 6DoF Tracking from Photometric Map

- Probabilistic **6DoF motion estimation** from $\pm C = -\nabla L \cdot \boldsymbol{u}$
- Assumes **photometric map** $(x, y, z$, grayscale Intensity) is **given**
- Useful for **VR/AR applications** (low-latency, HDR, no motion blur)
- Can run in **real time on a GPU**

Combining Standard Cameras with Event Cameras



Event Camera

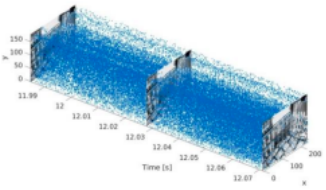
Standard Camera

Update rate	High (asynchronous): 1 MHz	Low (synchronous)
Dynamic Range	High (140 dB)	Low (60 dB)
Motion Blur	No	Yes
Static motion	No (event camera is a high pass filter)	Yes
Absolute intensity	No (but reconstructable up to a constant)	Yes
Maturity	< 10 years of research	> 60 years of research!

3

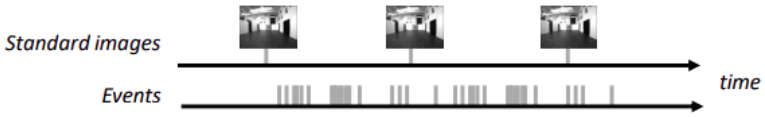
DAVIS sensor: Events + Images + IMU

- Combines an **event and a standard camera** in the **same pixel array** (→ the same pixel can both trigger events and integrate light intensity).
- It also has an **IMU**



Spatio-temporal visualization of the output of a DAVIS sensor

Temporal aggregation of events overlaid on a DAVIS frame



Application 1: Deblurring a blurry video

- **Idea**: A **blurry image** can be regarded as the **integral of a sequence of latent images** during the exposure time, while the **events** indicate the **changes between the latent images**
- **Solution**: sharp image obtained by subtracting the double integral of event from input image

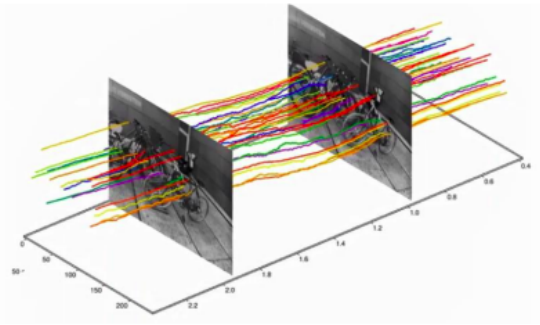
$$\log \quad - \iint \quad = \quad \log$$

Input blur image **Input events** **Output sharp image**

Application 3: Event-based KLT Tracking

- **Goal**: Extract **features from standard frames** and track them using only **events** in the **blind time between two frames**
- Uses the event generation model via joint estimation of patch warping and optic flow

Recap



- All the approaches seen so far use the **generative event model**

$$\log I(x, y, t + \Delta t) - \log I(x, y, t) = \pm C$$

- or its 1st order approximation

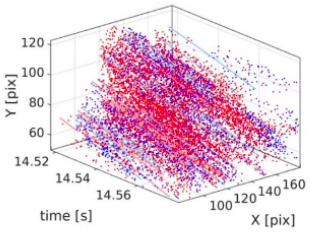
$$\pm C = -\nabla L \cdot \mathbf{u}$$

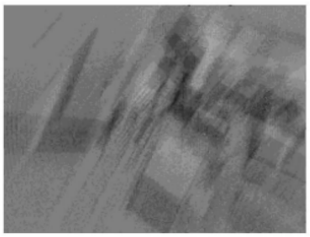
which **requires knowledge of the contrast sensitivity** C

- Unfortunately, $\boldsymbol{C}$ **is scene dependent** and might **differ from pixel to pixel**
- **Alternative approach: Contrast maximization framework**

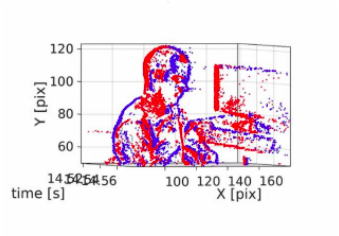
Contrast Maximization Framework

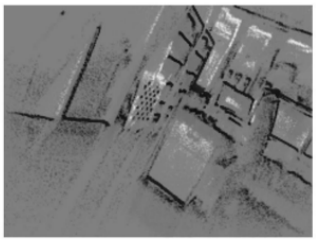
Idea: Warp spatio-temporal volume of events to **maximize contrast** (e.g., sharpness) of the resulting image





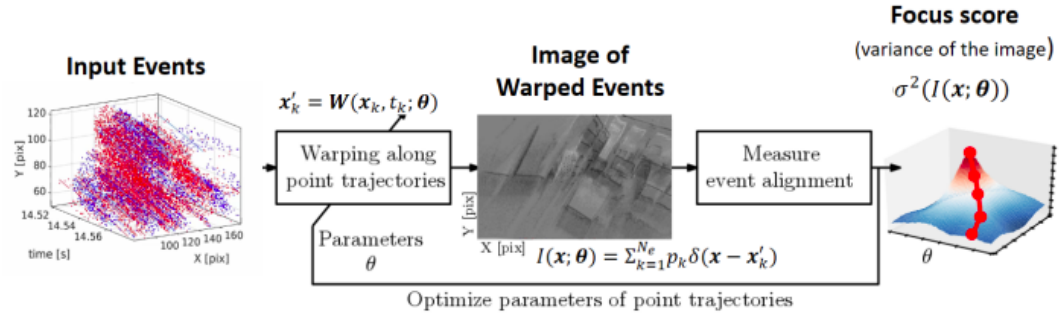
Aggregated image
without motion correction





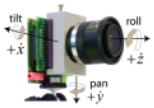
Aggregated image
with motion correction

Contrast Maximization Framework



- $x'_k = W(x_k, t_k; \theta)$: This warps the (x, y) pixels coordinates of each event, not their time. Possible warps: roto-translation, affine, homography.
- $I(x;\theta) = \Sigma_{k=1}^{N_e} p_k \delta(x - x'_k)$: This builds a grayscale image, where the intensity of each pixel at the warped location (x', y') is equal to the summation of the polarity p (i.e., positive and negative events $(+1, -1)$)
- $\sigma^2(I(x;\theta))$: The assumption here is that if an image contains **high variance** then there is a wide **spread of responses, both edge-like and non-edge like**, representative of a normal, in-focus image. But if there is **very low variance**, then there is a tiny spread of responses, indicating there are very little edges in the image. As we know, the more an image is blurred, the less edges there are.

Application 1: Image Stabilization



- Goal: **Estimate rotational motion (3DoF)** of an event camera
- Can process millions of events per second in real time on a smartphone PC (e.g., OdroidXU4)
- Works up to over ~1,000 deg/s

Application 2: Motion Segmentation

Application 3: Dynamic Obstacle Avoidance

- Works with relative speeds of up to **10 m/s**
- Perception **latency: 3.5 ms**

Application 4: "Ultimate SLAM"

Goal: combining **events**, **images**, and **IMU** for robust visual SLAM in HDR and high speed scenarios

Front End:
Feature tracking from Events and Frames

→

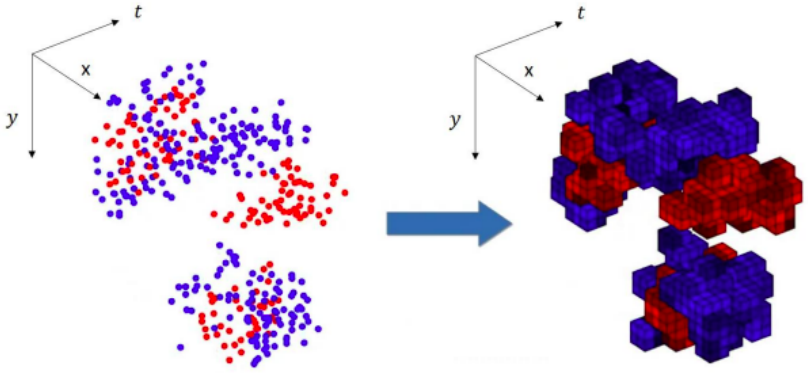
Back-End
State-of-the-art
Non-linear-optimization-based VIO

85% accuracy gain over standard VIO in HDR and high speed scenarios → enable autonomous navigation in low light

Input representation

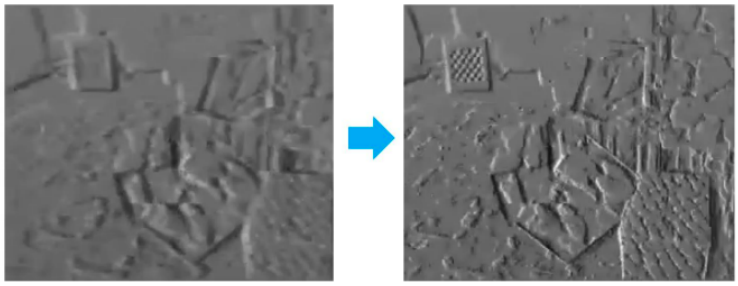
How do we pass sparse events into a convolutional neural network designed for standard images?

Represent events in space-time into a 3D voxel grid (x, y, t): each voxel contains sum of positive and negative events falling within the voxel



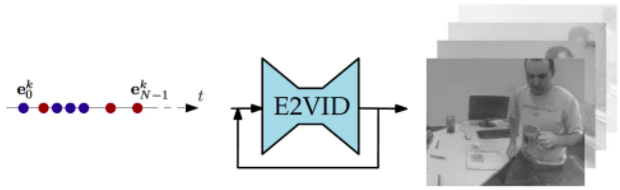
Contrast as Loss Function for Unsupervised Learning

- **Idea**: maximize sharpness of the aggregated event image



Overview

- **Recurrent neural network** (main module: Unet)
- Input: sequences of *event tensors* (3D spatio-temporal volumes of events[3])
- **Trained in simulation only**, without seeing a single real image
- To improve robustness **we randomize the contrast sensitivity** during simulation.
- Event camera simulator (ESIM): http://rpg.ifi.uzh.ch/esim.html



Applications:
1. Unsupervised learning of optical flow → ideas is to maximize the sharpness of the aggregated event image

2. **Image reconstruction** from events → see next slide

3. **Slow motion video**: we can combine an event camera with an HD RGB camera. We use events to upsample low-frame rate with low memory footprint