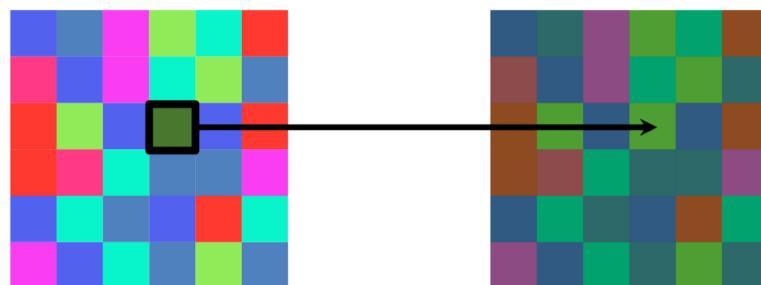


Image Processing

Point Operation



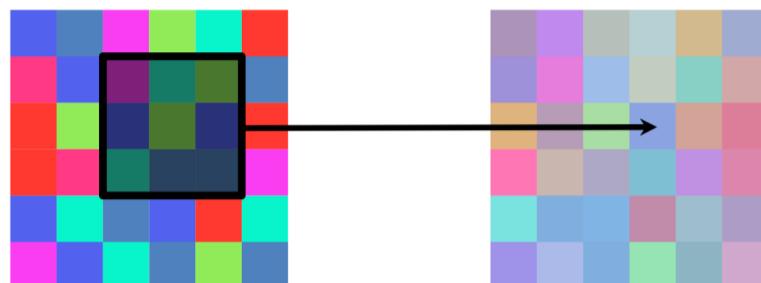
point processing →

Example:

$$g(x,y) = f(x,y) + 20 \text{ (brightness)}$$

$$g(x,y) = f(-x,y) \text{ (Horizontal Flip)}$$

Neighborhood Operation



"filtering" →

Form a new image whose pixel values are a combination of the original pixel values. (To get useful information from images)

Gamma Correction: Transformation non lineare applicata ai valori di intensità (luminosità) di un'immagine, per correggere la percezione visiva o l'adattamento ai dispositivi di visualizzazione.

Si usa per due motivi principali: 1) Percezione umana, l'occhio umano non percepisce le luminosità linearmente. 2) Hardware, i dispositivi non trattano le luminosità in modo lineare. Serve a correggere l'immagine per far sì che l'aspetto visivo sia naturale sullo schermo.

$$V_{out} = A V_{in}^\gamma$$

A: normalization constant

$\gamma > 1$: Reduce the brightness

$\gamma = 2.2$ common value

$\gamma < 1$: Increase brightness

Histogram Equalization: adjust the contrast of an image by using its histogram

Calculate the Histogram: $p(r_k) = \left(\frac{\# \text{ pixels with intensity } r_k}{\# \text{ pixels}} \right)$

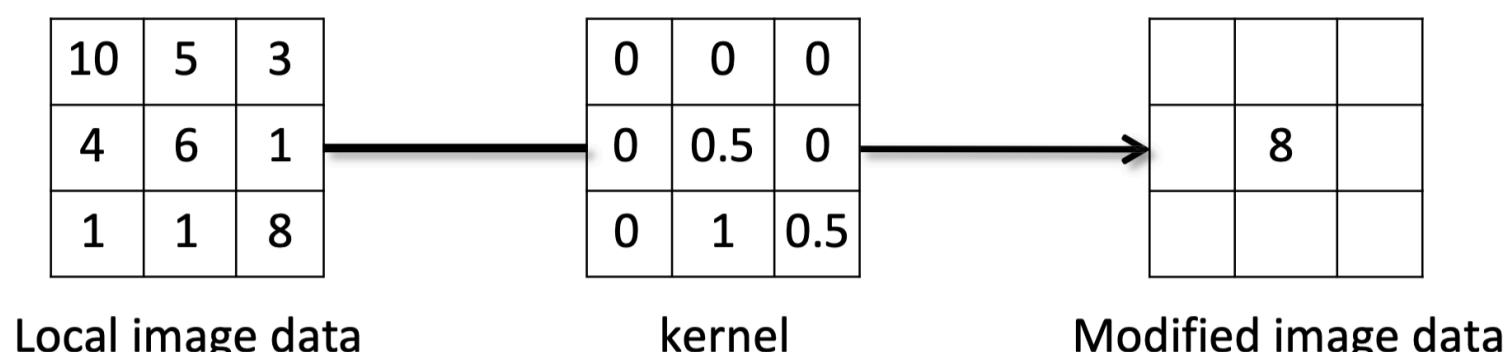
Normalize the Histogram: $CDF(r_k) = \sum_{i=0}^k p(r_i)$

Map Intensities : $r'_k = \text{round} \left(\frac{(CDF(r_k) - CDF_{min}) \cdot (L-1)}{N - CDF_{min}} \right)$

L: max pixel intensity

N: total number of pixels

Linear Filtering: Replace each pixel by a linear combination (a weighted sum) of its neighbors. The prescription for the linear combination is called the "kernel".



Mean Filtering: Serve a ridurre il rumore, ammorbidente variazioni locali e ottenere un'immagine visivamente più "liscia".

$$\text{Kernel : } \frac{1}{3} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Si fa scorrere queste finestre su ogni pixel dell'immagine.

Per ogni finestra, si leggono i pixel all'interno della finestra / si calcola la media aritmetica di quei valori e si assegna la media al pixel centrale.

Gaussian filter: Molto simile come scopo al mean filter, ma invece di fare la mediana semplice, fa una media pesata, dove i pixel più vicini al centro della finestra hanno un peso maggiore. Il peso di pixel segue una distribuzione gaussiana

Il peso per ogni pixel (i, j) nella finestra si calcola come:

$$G(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{(i^2+j^2)}{2\sigma^2}}$$

σ controlla quanto "larghe" è la campana \rightarrow più è grande più sfocatura

Sharpening: Serve a rinforzare i contorni e i dettagli fini di un'immagine. Funziona esattamente all'opposto dello smoothing:

- Lo smoothing attenua le variazioni locali
- Lo Sharpening amplifica le variazioni locali

$$\text{Sharpened Image} = \text{Original Image} + \alpha \cdot (\text{Original Image} - \text{Blurred Image})$$

Original - Blurred \rightarrow estrae i dettagli (also called high-pass filter)

$\alpha \rightarrow$ controlla l'intensità dello Sharpening

Bilateral Filtering: serve a levigare l'immagine, ma senza rovinare i bordi, e differenza di Gaussian e Mean.

Il bilateral filter tiene conto di due distanze contemporaneamente:

- Distanza spaziale (come il Gaussian Noise)
- Distanza di intensità: se il pixel ha un'intensità simile al centro pesa di più. Se l'intensità è molto diversa (bordi, contorni) pesa di meno

$$h[m, n] = \frac{1}{W_{mn}} \sum_{k, l} g[k, l] r_{mn}[k, l] f[m+k, n+l]$$

↓ ↓
Spatial Weighting Intensity range
 weighting

Image derivatives:

Le derivate di un'immagine indicano quanto cambia l'intensità di pixel nello spazio

$$\text{Derivata orizzontale: } I_x = \frac{\partial I}{\partial x}$$

$$\text{Kernel Classico: } [-1, 0, +1]$$

$$\text{Derivata verticale: } I_y = \frac{\partial I}{\partial y}$$

$$\text{Kernel Classico: } [-1, 0, +1]^T$$

Gradiente $\nabla I = (I_x, I_y)$: misura quanto cambia l'intensità in un punto

Magnitudo $= \sqrt{I_x^2 + I_y^2}$: dice quanto forte sta cambiando l'immagine in quel punto

Direzione del Gradiente $\theta = \arctan\left(\frac{I_y}{I_x}\right)$: dice in quale direzione c'è il massimo cambiamento di intensità

Prinz di calcolare le derivate conviene sempre applicare un filtro di smoothing. Per evitare di fare troppi passaggi, grazie alle proprietà delle convoluzioni, possono calcolare le derivate del kernel e poi applicarle all'immagine. (Derivative of Gaussian filter)

Laplace filter: è un operatore che calcola le seconde derivate spaziali di un'immagine. Evidencez zone dove l'intensità cambia rapidamente (bordi e texture). Molto sensibile al rumore

\rightarrow LoG: prima applico un filtro Gaussian poi applico il filtro Laplaciano. Usatissimo in edge detection.
("zero crossing" at edges)

Sobel Operator: common approximation of Derivative of Gaussian.

$$\frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

s_x

$$\frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

s_y

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Sobel filter

$$= \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Blurring

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

1D derivative filter

Canny Edge Detector:

- 1) Noise reduction: sfocia l'immagine con un Gaussian Filter
- 2) Calcolo del gradiente $\nabla I = (I_x, I_y)$
- 3) Magnitudo e direzione del gradiente
- 4) Non-maximum Supression

Dopo aver calcolato il gradiente (magnitudo), il bordo risulta "spesso": più pixel vicini hanno valori alti. Si mantiene solo il massimo locale per assottigliare i bordi. Per ogni pixel guarda il valore del magnitudo del Gradiente $M(x,y)$, controlla nella direzione del gradiente. Se $M(x,y)$ non è il maggiore di entrambi i vicini allora sopprime. (metti a zero), altrimenti mantieni.

5) Hysteresis Thresholding

Si usano due soglie:

- Threshold alto \rightarrow bordi fatti
- threshold basso \rightarrow bordi deboli

Il pixel viene accettato come bordo se:

- è sopra la soglia alta
- è sopra la soglia bassa e connesso ad un bordo forte

Gaussian Pyramid: Una sequenza di versioni sempre più piccole e sfocate dell'immagine originale

Ad ogni livello:

- Applichiamo Gaussian Filter (blurring)

- Poi subsampling

Serve per Analisi multi-scala, image blending (transizioni dolci tra immagini) e compressione immagini.

Non si può ottenere l'immagine iniziale dai livelli sottostanti

Laplacian Pyramid: una piramide che contiene solo i dettagli persi a ogni livello di smoothing.

Si costruisce partendo dalla Gaussian Pyramid:

A ogni livello Laplacian = livello corrente - livello successivo upsampleato

A partire dalla Laplacian Pyramid puoi ricostruire esattamente l'immagine originale

$$L_k = G_k + \text{Upsample}(G_{k+1})$$

Fourier transform

La trasformata di Fourier ti permette di guardare un'immagine non per quello che appare, ma per come è fatta dentro.

Cioè:

- Nello spazio dei pixel vedi cosa c'è nell'immagine: colori, forme, oggetti
- Nello spazio delle frequenze vedi come è fatta l'immagine (cambiamenti rapidi o lenti)

Prendiamo un'immagine $I(x,y)$

Là trasformiamo nello spazio delle frequenze: $F(u,v)$

Quello che otteniamo $F(u,v)$ non è semplicemente un'immagine di numeri reali, ma:

$$F(u,v) = \underbrace{A(u,v)}_{\text{Ampiezza}} \cdot e^{\underbrace{j\phi(u,v)}_{\text{Fase}}}$$

$A(u,v)$: ti dice quanto forte è presente quella frequenza nello spettro

Frequenze basse \rightarrow regioni uniformi

Frequenze alte \rightarrow dettagli: fini, bordi, texture

$\phi(u,v)$: codifica la posizione spaziale delle strutture nell'immagine
Là fase ti dice dove sono messe le cose nell'immagine

Ampiezza + Fase: ricostruzione perfetta dell'immagine

DFT: versione discinta e computabile della trasformata di Fourier.

Prendi un segnale finito e lo scomponi in una somma di onde sinusoidali di varie frequenze

$$F(u,v) = \sum \sum I(x,y) \cdot e^{-j2\pi \left(\frac{ux}{N} + \frac{vy}{M} \right)}$$

Immagine $N \times M$

Quando applichi un filtro ad un'immagine nello spazio dei pixel, in realtà stai manipolando il contenuto in frequenza dell'immagine

Convolution theorem

Convolution in spatial domain = multiplication in frequency domain

$$F[g * h] = F[g]F[h]$$

Low-pass filter : lascia passare solo le basse frequenze

Effetto sull'immagine:

- Sfocatura
- Rimuove dettagli fini
- Rende l'immagine più morbida
- Filtra il rumore ad alto frequenza

Maschera: Cerchio bianco al centro

High-pass filter : elimina le frequenze basse. Lascia passare solo le alte frequenze

Effetto sull'immagine:

- Evidenzia i bordi
- Enfatizza i cambiamenti rapidi
- Aumenta il contrasto sui dettagli

Maschera: Cerchio nero al centro (bordi bianchi)

Teorema di Campionamento di Nyquist - Shannon:

Il teorema ci dice a quale frequenza minima devi campionare un segnale continuo per poterlo ricostruire perfettamente.

Un segnale continuo con massima frequenza f_{\max} deve essere campionato almeno:

$$f_s \geq 2 \cdot f_{\max}$$

Feature extraction

Global features: Descrivono l'immagine intera (visione d'insieme). Si calcolano considerando l'intera immagine come un unico oggetto.

Local features: Si calcolano su piccole regioni dell'immagine (patch, punti chiave, pixel vicini). Sono robuste a cambiamenti globali e catturano dettagli locali (er. punti, bordi, texture)

Harris Corner Detector:

1- Compute Gaussian derivatives at each pixel

2- Compute second moment matrix H in a Gaussian window around each pixel

$$H = \sum_{(x,y) \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

3- Compute corner response function f or R

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{\det(H)}{\text{trace}(H)} \quad \text{or} \quad R = \lambda_1 \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2 = \det(H) - k \cdot \text{tr}(H)^2$$

4- Threshold f or R

5- Find local maxima of response function (non maximum suppression)

Harris detector is:

- equivariant to translation and rotation
- partially invariant to affine intensity change
- neither invariant nor equivariant to scaling

Scale space: Prendo l'immagine e creo tante versioni sfocate, ognuna con un diverso grado di blur. Ogni versione sfocata corrisponde a una scala diversa.

Per ogni punto dell'immagine guardo la risposta* in tutte le scale. Se il punto è un "corner forte" proprio a quella scala servirà un massimo locale. Questo mi dice sia dove è il keypoint, sia a quale scala appartiene (Analisi tridimensionale)

* Scale-normalized Laplacian of Gaussian filter

$$\nabla_{\text{norm}}^2 = \sigma^2 \left(\frac{\partial^2}{\partial x^2} g + \frac{\partial^2}{\partial y^2} g \right)$$

In alternativa allo scale-space continuo ottenuto variando σ , posso costruire una Gaussian Pyramid, riducendo progressivamente l'immagine, e applicare i detector sempre con finestre fisse

Feature descriptor

È una rappresentazione numerica che cattura il contenuto visivo attorno al keypoint, in modo invariante a trasformazioni. Serve per confrontare due immagini, per capire se due keypoint in immagini diverse corrispondono.

HOG descriptor: descrive l'immagine basandosi su come sono distribuite le direzioni dei bordi, localmente.

Preprocessing: Si seleziona una finestra rettangolare sull'immagine. Poi si ridimensiona sempre a 64×128 pixel, indipendentemente dalla dimensione originale. Così tutte le immagini hanno sempre la stessa dimensione in input. Il descriptor HOG risultante avrà sempre le stesse lunghezze.

Procedimento

1) Calcolo del gradiente

Primo calcolo i gradienti orizzontali e verticali $I_x \times I_y$.

Poi orientazione del gradiente e magnitudo.

2) Divisione in celle

L'immagine viene divisa in piccole celle 8×8

3) Costruzione dell'istogramma per celle

Per ogni cella:

- Si costruisce un istogramma con i contributi delle direzioni di gradiente
- Esempio: 8 bins \rightarrow angoli da 0° a 180° , ogni bin copre 20°

I gradienti con magnitudo maggiore contano di più nel bin.

4) Normalizzazione con blocchi

Le celle vengono raggruppate in blocchi (2×2 celle)

Si normalizzano i vettori per rendere il descriptor robusto ai cambiamenti di illuminazione

5) Concatenazione del vettore finale

Tutti gli histogrammi normalizzati vengono concatenati in un unico vettore feature

HOG è pensato per descrivere un'intera finestra o regione definita; non serve keypoint detection

SIFT:

1) Costruzione dello scale-space

Parto dall'immagine originale $I(x,y)$

Crea una serie di immagini sfocate applicando Gaussian Blur

2) Ottavo

Per notivi di efficienza, dopo aver generato alcune immagini sfocate su scale corrente:

- Downsample l'immagine (riduci la dimensione di 2x)
- Ripeti a generare nuove scale sulla versione ridotta

Ogni ottavo rappresenta un nuovo livello di zoom-out

3) Calcolo del Difference of Gaussians (DoG)

Tra ogni coppia di immagini sfocate successivamente si calcola:

$$\text{DoG}(x, y, \sigma) = L(x, y, \sigma_{in}) - L(x, y, \sigma_{out})$$

Il DoG funziona come approssimazione efficiente del Laplacian of Gaussian.

4) Keypoint Detection

Per ogni pixel in ogni DoG lo si confronta con i suoi 26 vicini. Se il valore è il massimo tra tutti questi \rightarrow possibile Keypoint

5) Refinement e Filtraggio di Keypoints

Vengono eliminati quelli con basso contrasto e su bordi allungati.

6) Assegnazione dell'orientazione

Per ogni keypoint mantenuto:

- Prendo una finestra attorno al Keypoint (di raggio proporzionale a σ)
- Per ogni pixel nella finestra calcolo Magnitudo e Orientazione del gradiente
- Costruisco un istogramma di orientazioni (tipicamente 36 bins da 10° l'uno) considerando anche il peso del magnitudo
- Trovo il massimo dell'istogramma \rightarrow orientazione dominante del keypoint

Così ottengo invarianza alla rotazione

7) Creazione del feature descriptor

- Prendo una patch 16×16 centrata sul keypoint
- Applico una rotazione per riallineare la patch all'orientazione dominante (invarianti alla rotazione)
- La patch 16×16 viene suddivisa in 16 celle $= 4 \times 4$
- Per ogni pixel della patch calcolo

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

$$\theta(x,y) = \arctan\left(\frac{I_y}{I_x}\right) - 90^\circ \quad \rightarrow \text{misuro i gradienti nel sistema ruotato}$$
- In ogni cella 4×4 creo un istogramma delle direzioni del gradiente & bin da 45° ciascuno
- (Quindi ogni cella produce un vettore di 8 valori)
- Concateno tutti gli istogrammi: 16 celle \times 8 valori ciascuno = 128 dimensioni

SURF (Speeded-UP Robust Features) :

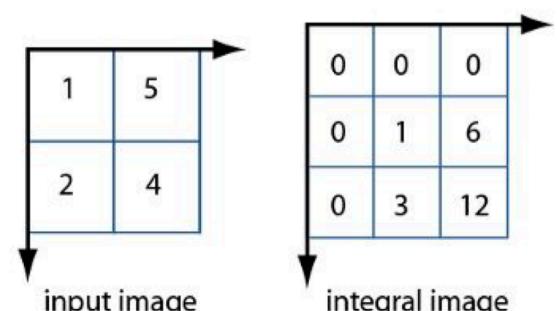
SURF interest points are in-plane rotation-invariant, robust to noise, and overall, extremely fast to calculate.

Three steps:

1. Interest Point Detection
2. Interest Point Description
3. Interest Point Matching

Integral Images are an image transformation such that any entry of an integral image $I_S(x)$ at a location $x=(x,y)^T$ represents the sum of all pixels in the input image I within a rectangular region formed by the origin and x

(Serve per calcolare i filtri box molto rapidamente)



The detector detects blob-like structures at locations where the determinant of the Hessian is maximum

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{yx}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix}$$

$L_{xx}(x, \sigma)$ is the convolution of the Gaussian second order derivative with the image I in point x

The Hessian can be approximated using box filters.

$$\det(H_{\text{approx}}) = D_{xx} D_{yy} - (w D_{xy})^2$$

SURF invece di cambiare il grado di sfocatura (come SIFT), cambia la dimensione del filtro che applica all'immagine. Più il filtro è grande, più guarda un'area ampia \rightarrow quindi simula scale maggiori.

Evita di calcolare convoluzioni gaussiane lente

Assegnazione dell'orientazione:

- Consideriamo una regione circolare attorno al keypoint: raggio: $6s$: scala alla quale è stato trovato il keypoint
- Calcoliamo i gradienti: SURF non usa i gradienti classici, usa dei Haar filters (filtri molto veloci). Ottiene due valori per ogni pixel nel cerchio dx e dy
- Non tutti i pixel contano allo stesso modo, i pixel più vicini al centro contano di più \rightarrow vengono pesati tramite una Gaussian centered sul keypoint
- Inseguiamo di dividere il cerchio in settori di 60° . Per ogni settore sommo tutti i dx e dy dei pixel che cadono in quella direzione. Per ogni finestra angolare ottieni: somma vettoriale = $(\sum dx, \sum dy)$
- Guardo tutte le finestre e prendo quelle dove il vettore somma è più lungo. La direzione di questo vettore diventa la orientazione dominante

Interest point Description:

- Prende un quadrato di dimensione 20s centrato sul keypoint
- Dividiamo il quadrato in una griglia di quadrati 4×4 (16 celle)
- Per ogni cella, SURF calcola con Haar wavelets, quanto cambia l'immagine orizzontalmente (d_x) e verticalmente (d_y)

$$V = \begin{bmatrix} \sum d_x \\ \sum d_y \\ \sum |d_x| \\ \sum |d_y| \end{bmatrix}$$

Analizza 25 punti equidistanti:
local feature vector

I 16 local feature vector vengono concatenati per ottenere un feature vector da 64 elementi

Binary descriptor: è un feature descriptor che non restituisce un vettore di numeri reali (come SIFT o SURF), ma restituisce un semplice sequenza di bit: 0 e 1

Più compatti e veloci da confrontare

Procedimento

- Si rileva un key point e si prende una patch attorno a esso
- Si fanno dei test binari su coppie di pixel: Confronta due pixel nella patch: è più luminoso il primo o il secondo
Se il primo è più luminoso \rightarrow altrimenti 0
- Si eseguono molti di questi confronti (tipicamente 256 o più)
Ogni confronto produce 1 bit \rightarrow si ottiene così un vettore

Binario

BRIEF: binary descriptor puro ↑

ORB: Un'estensione di BRIEF che calcola l'orientazione per ogni keypoint per diventare rotation-invariant e sceglie le migliori coppie possibili

Feature Matching:

1. Define distance function that compares two descriptors
Any distance metric would work: Mean squared Error e Mean Absolute Error loss are commonly used
2. Test all the features in I_2 , find the one with min distance or find top K matches

Better approach: distance ratio: $\frac{\|f_1 - f_2\|}{\|f_1 - f_2'\|}$

f_2 is best SSD match to f_1 in I_2

f_2' is 2nd best SSD match to f_1 in I_2

Sorting by this ratio puts matches in order of confidence

set the distance ratio threshold (ρ) to around 0,5, which means that we require our best match to be at least twice as close as our second best match to our initial features descriptor

2D Transformation and Alignment

- **Image Filtering:** change range of image: $g(x) = h(f(x))$
modificano i valori dei pixel, quanto è chiaro
o scuro ogni pixel
- **Image Warping:** change domain of image: $g(x) = f(h(x))$
Col warping modifichi dove sta ogni pixel

Linear Transformations: Trasformazioni che possono essere rappresentate da una matrice 2×2

• Rotation by angle: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

• 2D Shear : $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

• 2D Mirror about Y axis : $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

• 2D Mirror over (0,0) : $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

• 2D Scale Around (0,0) : $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Properties:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under Composition

Affine Transformations: any transformation represented by a 3×3 matrix with last row $[0 \ 0 \ 1]$ we call an affine transformation

Translation : $\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$
 \hookrightarrow add a 3rd coordinate

Properties of affine transformations:

- Origin does not necessarily map to origin

Projective Transformation: A homography is a transformation that maps point from one plane to another

L'omografia si rappresenta con una matrice 3×3 sulle coordinate omogenee:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Incluse tutte le trasformazioni affini e molte più gestive anche la prospettiva

Properties of projective transformations:

- Origin does not necessarily map to origin
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved

Computing 2D transformations

Translations:

Displacement of match $i = (x'_i - x_i, y'_i - y_i)$

$$(x_t, y_t) = \left(\frac{1}{n} \sum_{i=1}^n x'_i - x_i, \frac{1}{n} \sum_{i=1}^n y'_i - y_i \right)$$

$$\begin{aligned} x_i + x_t &= x'_i \\ y_i + y_t &= y'_i \end{aligned} \quad \left\{ \begin{array}{l} \text{If you have more than two points, you can use the} \\ \text{method of least squares to obtain a more robust} \\ \text{estimation of the translation parameters} \end{array} \right.$$

Least Squares find the translation that minimizes the sum of squared errors between the corresponding points, by computing the average displacement

$$r_{xi}(x_t) = (x_i + x_t) - x'_i$$

$$r_{yi}(y_t) = (y_i + y_t) - y'_i$$

$$t = (A^T A)^{-1} A^T b$$

$$C(x_t, y_t) = \sum_{i=1}^n (r_{xi}(x_t))^2 + r_{yi}(y_t)^2$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x'_i - x_i \\ y'_i - y_i \\ \vdots \end{bmatrix}$$

Affine transformation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$r_{xi}(a, b, c, d, e, f) = (ax_i + by_i + c) - x'_i$$

$$r_{yi}(a, b, c, d, e, f) = (dx_i + ey_i + f) - y'_i$$

$$C(a, b, c, d, e, f) = \sum_{i=1}^n (r_{xi}(a, b, c, d, e, f))^2 + r_{yi}(a, b, c, d, e, f)^2$$

Homographies:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ & & & & & : & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A}_{2n \times 9}$$

$$\mathbf{h}_9$$

$$\mathbf{0}_{2n}$$

Defines a least squares problem: minimize $\|\mathbf{Ah} - \mathbf{0}\|^2$

RANSAC: is an iterative algorithm that randomly selects subsets of data to estimate a model, evaluates the number of inliers for each model, and keeps the one with the largest consensus set, making it robust to outliers

Step 1: Randomly choose s samples

Select randomly a subset of s data points

s is the minimum number of points required to fit the model

Step 2: Fit a model to those samples

Step 3: Count the number of inliers (through a threshold)

Step 4: Repeat N times

Step 5: Choose the model with the largest set of inliers

Warping: Send each pixel (x, y) to its corresponding location

$$(x', y') = T(x, y) \text{ in } g(x', y').$$

If pixel lands "between" two pixels? Add contribution
to several pixels, normalize later (splatting)

Can still result in holes

Inverse Warping: Get each pixel $g(x', y')$ from its corresponding
location $(x, y) = T^{-1}(x', y')$ in $f(x, y)$

What if pixels comes from "between" two
pixels? Resample color value from interpolated
source image. No holes

Optical Flow

Optical flow is the apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. Is used to see how every point is moving frame to frame in a video sequence.

Motion Field: representing the projection of the 3D motion of points in the scene onto the image plane.

Can be the result of camera motion or object motion

Optical Flow \neq Motion Field

So:

Motion Field = the true motion of the scene in 3D world, projected onto the image plane

Optical Flow = what we observe as the displacement of pixels between two images.

If we know the image motion we can compute images at intermediate frames.

Aperture Problem: Quando osserviamo il movimento guardando solo attraverso una finestra piccola, in molti casi non possiamo determinare l'intero vettore di movimento. Possiamo osservare solo il componente del moto perpendicolare al bordo (gradienze dell'intensità)

Equazione di brightness constancy

$$I_x \cdot u + I_y \cdot v + I_t = 0$$

I_x : Derivate spaziale lungo x

u : movimento orizzontale

I_y : Derivate spaziale lungo y

v : movimento verticale

I_t : Derivate temporale

Questa formula lega il movimento (u, v) con i gradienti dell'immagine

Problema: C'è solo un'equazione per due incognite

Vuol dire che con questa formula puoi sapere solo una combinazione di u e v

Aperitur Problem \rightarrow immagine cambia intensità solo in una direzione ($I_y = 0 \vee I_x = 0$)

Key Assumptions

- Brightness constancy: projection of the same point looks the same in every frame ①
- Small motion: points do not move very far ②
- Spatial coherence: points move like their neighbours

$$I(x(t), y(t), t) = C \text{ (constant)} \quad ①$$

$$I(x + u s t, y + v s t, t + s t) = I(x, y, t) \quad ② \text{ For small space-time step}$$

$$\hookrightarrow \cancel{I(x, y, t)} + \frac{\partial I}{\partial x} s x + \frac{\partial I}{\partial y} s y + \frac{\partial I}{\partial t} s t = \cancel{I(x, y, t)}$$

Taylor series expansion

$$\text{divid by } s t : \frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

Brightness
constancy
Equation

Lucas - Kanade Optical Flow: Assume that the surrounding patch has "constant flow". For each pixel assume motion field, and hence optical flow (u, v) , is constant within a small neighborhood

Using 5×5 image patch, gives us 25 equations

$$I_x(p_1)u + I_y(p_1)v = -I_t(p_1)$$

$$I_x(p_2)u + I_y(p_2)v = -I_t(p_2)$$

:

:

:

2 unknown: find least square solution

$$I_x(p_{2s})u + I_y(p_{2s})v = -I_t(p_{2s})$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{2s}) & I_y(p_{2s}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{2s}) \end{bmatrix}$$

$A: n^2 \times 2$

$u: 2 \times 1$

known

$B: n^2 \times 1$

known

$$u = (A^T A)^{-1} A^T B$$

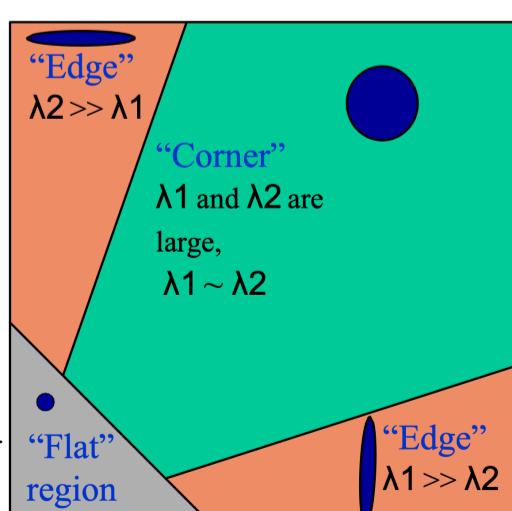
$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \quad \begin{array}{l} \text{should be invertible} \\ \det \neq 0 \end{array}$$

should be well conditioned

$$A^T B = \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ is the second moment matrix (Harris Corner detector)

\hookrightarrow Eigen vectors and eigenvalues of $A^T A$ relate to edge direction and magnitude



Corner and high texture regions are when λ_1, λ_2 are big; this is also when Lucas-Kanade optical flow works best.

Corner and high texture regions are regions with two different directions of gradient

Thus, the matrix $A^T A$ summarizes gradient information in the window. Its eigenvalues indicate if there is enough variation in both directions to reliably estimate optical flow; both eigen values must be large to avoid Aver. Problem

Coars-to-fine Flow Estimation: uses image pyramids to estimate optical flow. Large motion are first estimated at low resolution, then progressively refined at higher resolution where residual motion becomes small and linear model remain valid

Motion Segmentation: How do we represent the motion in this scene? Break image sequence into "layers" each of which has a coherent (affine) motion

1. Obtain a set of initial affine motion hypotheses

- Divide the image into blocks and estimate affine motion parameters in each block by least squares
 - Eliminate hypotheses with high residual error

2. Map into motion parameter space

3. Perform k-means clustering on affine motion parameters

- Merge clusters that are close and retain the largest clusters to obtain a smaller set of hypotheses to describe all the motions in the scene

4. Assign each pixel to best hypothesis --- iterate

Horn-Schunck Optical Flow: Considerate tutte l'immagine assieme

Aggiunge un vincolo di smoothness globale

↳ Assume che il campo d: flusso ottico sia liscio su tutta l'immagine, cioè i vicini si muovono in modo simile

Horn-Schunck trasforma il problema in una minimizzazione globale

$$E(u, v) = \iint (I_x u + I_y v + I_t)^2 + \lambda^2 (\|\nabla u\|^2 + \|\nabla v\|^2) dx dy$$

Primo Termine: brightness constancy

Secondo Termine: smoothness \rightarrow penalizza grandi variazioni locali del flux

Non esiste formula chiusa \rightarrow serve procedura iterativa

Algoritmo iterativo:

1) Pre-calcolo dei gradienti: I_x, I_y, I_t su tutta l'immagine

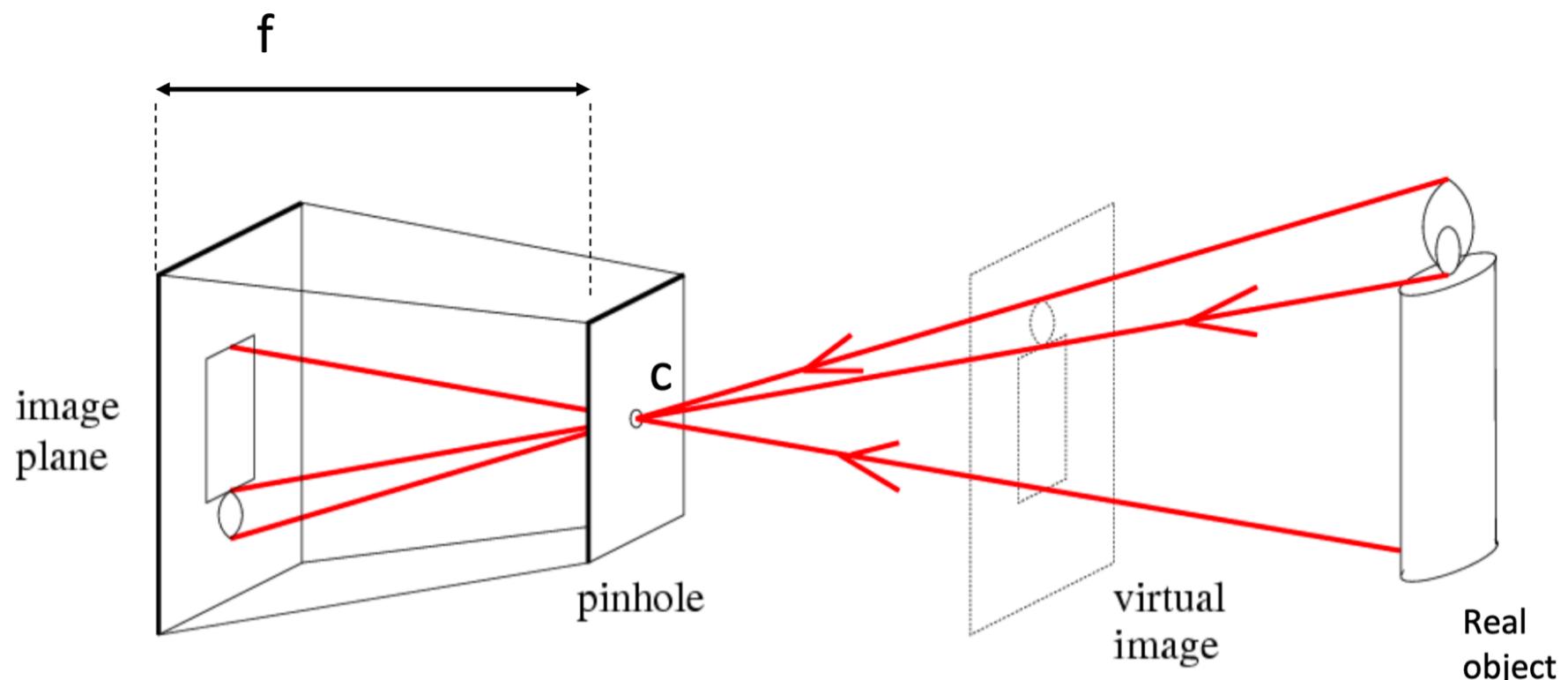
2) Inizializzazione: $u=0, v=0$, supponiamo inizialmente che non ci sia movimento

3) Calcolo delle medie dei vicini: $\bar{u}_{k1}, \bar{v}_{k1}$

4) Correzione per rispettare la brightness constancy (fino a che non converge)

Usa le formule di aggiornamento $\hat{u}_{k1} = \bar{u}_{k1} - \frac{I_x \bar{u}_{k1} + I_y \bar{v}_{k1} + I_t}{\lambda^2 + I_x^2 + I_y^2} I_x (I_y - \bar{v}_{k1})$

Camera Calibration



f = Focal length

c = Optical center of the camera

In perspective projection, 3D points in camera coordinates are mapped to the image plane by dividing them by their z component and multiplying with the focal length

$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \longrightarrow P = \begin{bmatrix} U \\ V \end{bmatrix} \quad U = -X \cdot \frac{f}{Z} \quad V = -Y \cdot \frac{f}{Z}$$

In order to do the reconstruction of a scene in 3D we need two things:

- position and orientation of the camera (external parameters of the camera)
- how the camera maps points in the world onto its image plane (internal parameters of the camera such as focal length)

So we need to estimate intrinsic and extrinsic parameters but first of all we need a model for the camera: projection matrix

From the projection matrix we can derive the internal and external parameters of the camera

World Coordinates	Camera Coordinates	Image Coordinates
$\mathbf{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$	$\mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$	$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$

In perspective projection with a pinhole camera, 3D points are mapped to the image plane by similar triangles, leading to projection equations:

$$x_i = x_o \cdot \frac{f}{z_o} \quad y_i = y_o \cdot \frac{f}{z_o} \quad P_o = (x_o, y_o, z_o) \text{ Punto nello spazio 3D}$$

First, 3D points are expressed in world coordinates. These are transformed to camera coordinates and then, the pinhole model project camera coordinates onto the image plane

Image Plane \rightarrow Image Sensor

The image plane is the continuous projection surface in mm used in perspective projection equations; the image sensor is the discrete pixel grid where the image is sampled and digitized.

$$u = m_x \cdot x_i + o_x \quad m_x, m_y \text{ pixel density}$$

$$v = m_y \cdot y_i + o_y \quad o_x, o_y \text{ shift by the principal point}$$

||

$$u = m_x f \cdot \frac{x_c}{z_c} + o_x = f_x \cdot \frac{x_c}{z_c} + o_x$$

$$v = m_y f \cdot \frac{y_c}{z_c} + o_y = f_y \cdot \frac{y_c}{z_c} + o_y$$

f_x, f_y

Intrinsic Parameters:

- f_x : lunghezza focale in pixel sull'asse X
- f_y : lunghezza focale in pixel sull'asse Y
- o_x : coordinate del principale punto lungo X (in pixel)
- o_y : coordinate del principale punto lungo Y (in pixel)

Lenses: Implement the same projection as pinhole cameras by bending light rays, allowing larger apertures to collect more light while maintaining perspective geometry

The focal length f is the distance from the lens to the focal point, where parallel rays of light converge

$$\frac{1}{i} + \frac{1}{o} = \frac{1}{f} \quad i = \text{image distance} \quad o = \text{object distance}$$

Equation for perspective projections are non linear \rightarrow homogeneous coordinates

$$u = (u, v) \quad \tilde{u} = (\tilde{u}, \tilde{v}, \tilde{w}) : \quad u = \frac{\tilde{u}}{\tilde{w}} \quad v = \frac{\tilde{v}}{\tilde{w}} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} 2D \rightarrow 3D \text{ point}$$

$$u = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{w}u \\ \tilde{w}v \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \tilde{u}$$

$$x = (x, y, z) \quad \tilde{x} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \quad \tilde{w} \neq 0 \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} 3D \rightarrow 4D \text{ point}$$

$$x = \frac{\tilde{x}}{\tilde{w}} \quad y = \frac{\tilde{y}}{\tilde{w}} \quad z = \frac{\tilde{z}}{\tilde{w}}$$

Homogenous Coordinates of (u, v) :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + o_x z_c \\ f_y y_c + o_y z_c \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Calibration Matrix:

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Upper Right Triangular Matrix}$$

Intrinsic Matrix:

$$M_{int} = [k|0] = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\tilde{x} = [k|0] \tilde{x}_c = M_{int} \tilde{x}_c$$

Position c_w and Orientation R of the camera in the world coordinate frame w are the camera's Extrinsic Parameters

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \rightarrow \begin{array}{l} \text{Row 1: Direction of } \hat{x}_c \text{ in world coordinate frame} \\ \text{Row 2: Direction of } \hat{y}_c \text{ " " " " " "} \\ \text{Row 3: Direction of } \hat{z}_c \text{ " " " " " "} \end{array}$$

$$R \text{ is Orthonormal} \rightarrow R^{-1} = R^T \quad R^T R = R R^T = I$$

Given the extrinsic parameters (R, c_w) , we can compute the camera-centric location of the point P in the world coordinate frame:

$$x_c = R(x_w - c_w) = Rx_w - Rc_w = Rx_w + t$$

$$x_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

$$\begin{bmatrix} \tilde{x}_c \\ \tilde{y}_c \\ \tilde{z}_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Extrinsic Matrix: M_{ext}

$$\tilde{x}_c = M_{ext} \tilde{x}_w$$

So

$$\tilde{\mu} = H_{int} \tilde{x}_c \quad \tilde{x}_c = H_{ext} \tilde{x}_w$$

$$\hookrightarrow \tilde{\mu} = H_{int} H_{ext} \tilde{x}_w = P \tilde{x}_w \quad (\text{Dal mondo all'immagine} \rightarrow \text{non il contrario})$$

P = projection Matrix

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Camera Calibration Procedure

Step 1 : Capture an image of object of known geometry
(x_w known)

Step 2 : Identify correspondences between 3D scene points
and image points ($\mu = [\begin{matrix} u \\ v \end{matrix}]$ known)

Step 3 : For each corresponding point i in scene and image

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_{wi} \\ y_{wi} \\ z_{wi} \\ 1 \end{bmatrix}$$

Expanding the matrix as linear equations :

$$u_i = \frac{p_{11}x_{wi} + p_{12}y_{wi} + p_{13}z_{wi} + p_{14}}{p_{31}x_{wi} + p_{32}y_{wi} + p_{33}z_{wi} + p_{34}}$$

$$v_i = \frac{p_{21}x_{wi} + p_{22}y_{wi} + p_{23}z_{wi} + p_{24}}{p_{31}x_{wi} + p_{32}y_{wi} + p_{33}z_{wi} + p_{34}}$$

Step 4 : Rearranging the terms

$$\begin{bmatrix} x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1 x_w^{(1)} & -u_1 y_w^{(1)} & -u_1 z_w^{(1)} & -u_1 \\ 0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1 x_w^{(1)} & -v_1 y_w^{(1)} & -v_1 z_w^{(1)} & -v_1 \\ \vdots & \vdots \\ x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & 0 & 0 & 0 & 0 & -u_i x_w^{(i)} & -u_i y_w^{(i)} & -u_i z_w^{(i)} & -u_i \\ 0 & 0 & 0 & 0 & x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & -v_i x_w^{(i)} & -v_i y_w^{(i)} & -v_i z_w^{(i)} & -v_i \\ \vdots & \vdots \\ x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_n x_w^{(n)} & -u_n y_w^{(n)} & -u_n z_w^{(n)} & -u_n \\ 0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_n x_w^{(n)} & -v_n y_w^{(n)} & -v_n z_w^{(n)} & -v_n \end{bmatrix} = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}$$

Step 5: Solve for p

$$A_p = 0$$

Since projection results are in homogeneous coordinates, any scalar multiple of the projection matrix produces the same image point after normalization, making the projection matrix defined only up to a scale.

Quindi vogliamo trovare una matrice P che soddisfi:

$$A \cdot p = 0$$

Poiché i dati reali sono riusciti non possono avere

$$A \cdot p = 0$$

Cevchiz =

$\min A_p \Rightarrow$ vogliamo che A_p sia il più vicino possibile
a zero

Option 1:

Fissiamo arbitrariamente uno degli elementi di P , ad esempio

$$P_{34} = 1$$

Option 2:

Normalizziamo il vettore p in modo che abbia norma unitaria

$$\|p\|^2 = 1$$

Scegliere più stabile numericamente

Il problema diventa $\min \|Ap\|^2$ tale che $\|p\|^2 = 1$

$$= \min (p^T A^T A p) \quad \text{con } p^T p = 1$$

Definiamo la Loss Function $L(p, \lambda)$:

$$L(p, \lambda) = p^T A^T A p - \lambda (p^T p - 1)$$

$$\text{Derivando } L(p, \lambda) : \quad 2A^T A p - 2\lambda p \Rightarrow A^T A p = \lambda p$$

Problema degli
Autovettori

p è l'autovettore di $A^T A$ corrispondente al più piccolo autovalore
minimo autovettore \rightarrow soluzione del least squares up to scale

Una volta ottenuto P possiamo ricavare le intrinsics matrix e
le extrinsics matrix

$$P = K \cdot [R | t]$$

Quindi P è formato da

una parte $M = K \cdot R \Rightarrow$ Ra factorization

, una parte $K \cdot t \Rightarrow t = K^{-1} p_4$

Triangulation using two cameras

l: left camera

r: right camera

$$u_l = f_x \frac{x}{z} + o_x$$

$$u_r = f_x \frac{x-b}{z} + o_x$$

$$v_l = f_y \frac{y}{z} + o_y$$

$$v_r = f_y \frac{y}{z} + o_y$$

b: Baseline \rightarrow the physical distance between the centers of projection of two cameras in a stereo setup

Solving for (x,y,z)

$$x = \frac{b(u_l - o_x)}{(u_l - u_r)} \quad y = \frac{bf_x(v_l - o_y)}{f_y(u_l - u_r)} \quad z = \frac{bf_x}{(u_l - u_r)}$$

$u_l - u_r$ is called Disparity \rightarrow is the difference in pixel positions of the same 3D point projected into the left and right images

Stereo block Matching (Template Matching)

1) Si Selezione un piccolo riguardo (window) nella foto di sinistra \rightarrow Template Window T

2) Si scansiona lungo la search scan line L orizzontale della foto d. destra. Orizzontale perché in una configurazione

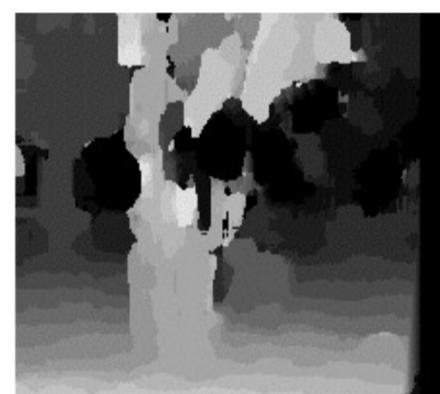
Stereo rettificata i corrispondenti sono sempre sulla stessa riga

3) Si confronta il Template T con tutte le finestre lungo la scanline, misurando la similarità.

Si possono usare:

- Sum of squared differences: $SSD(k,l) = \sum_{(i,j) \in T} |E_l(i,j) - E_r(i+k, j+l)|^2$
- Sum of Absolute differences: $SAD(k,l) = \sum_{(i,j) \in T} |E_l(i,j) - E_r(i+k, j+l)|$
- Normalized Cross Correlation: $NCC(k,l) = \frac{\sum_{(i,j) \in T} E_l(i,j) E_r(i+k, j+l)}{\sqrt{\sum_{(i,j) \in T} E_l(i,j)^2 \sum_{(i,j) \in T} E_r(i+k, j+l)^2}}$

How large should window be? Adaptive Window method: For each point, matching using windows of multiple sizes and use the disparity that is a result of the best similarity measure.



$W = 3$

$W = 20$

Smaller window

- + More detail
- More noise

Larger window

- + Smoother disparity maps
- Less detail
- Fails near boundaries

A disparity map assign to each pixel the horizontal shift between the left and right images in a stereo pair, and is directly related to depth via triangulation

Epipolar Geometry and Fundamental Matrix

Can we find the 3D scene point of a pixel?

3D to 2D : $u = f_x \frac{x_c}{z_c} + o_x$ $v = f_y \frac{y_c}{z_c} + o_y$
(point)

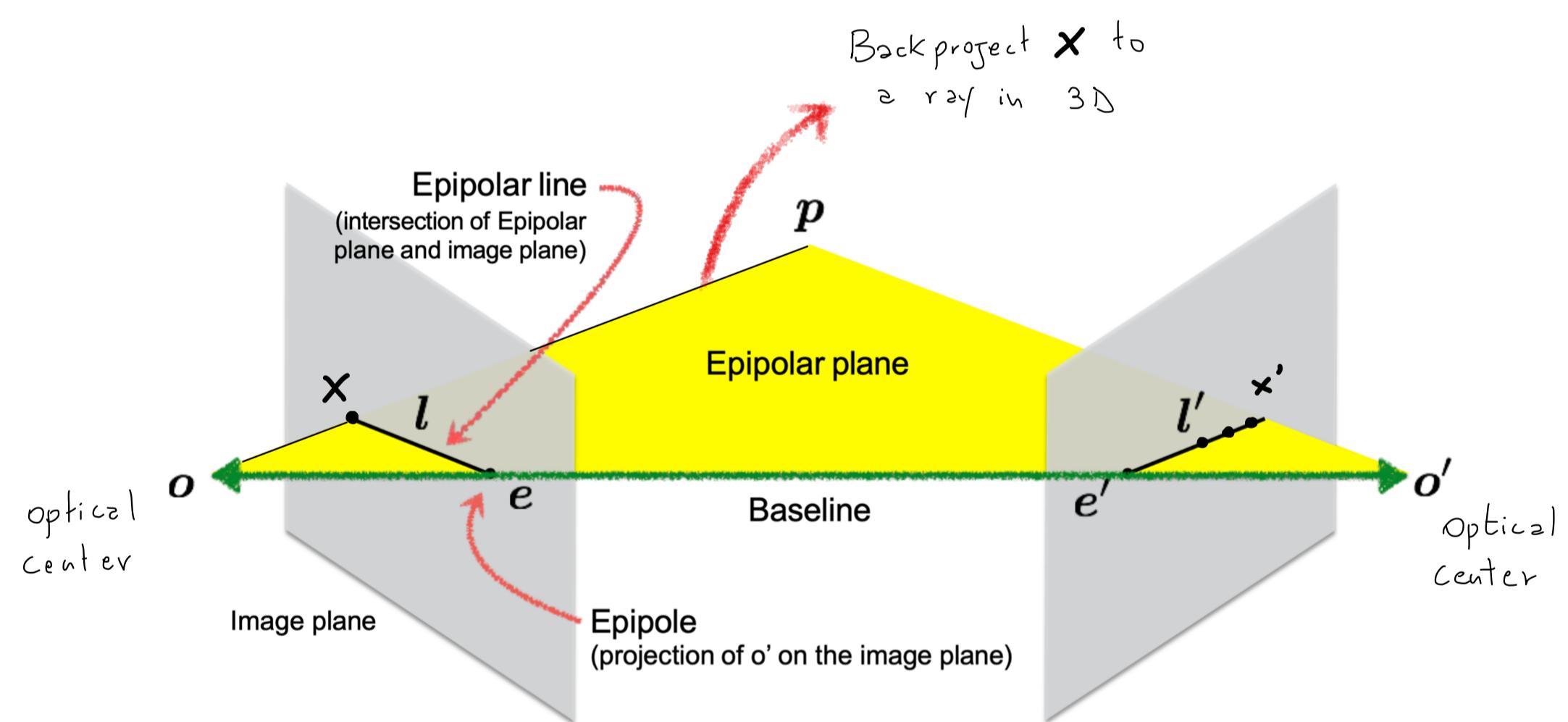
2D to 3D
(ray) : $x = \frac{z}{f_x} (u - o_x)$ $y = \frac{z}{f_y} (v - o_y)$ $z > 0$
back projection

Thus, we can't directly compute the 3D position because depth z_c is unknown.

What we can compute is a ray in 3D space that passes through that pixel

To fully recover the 3D position, at least two views are required, allowing triangulation of the intersection point of two rays

Epipolar Geometry



Potential matches for x lie on the epipolar line l' (epipolar constraint)

Obviously a different point in 3D will form a different epipolar plane and therefore different epipolar lines. But these epipolar lines go through epipoles as well. So Epipole lies where all epipolar lines meet

Epipolar constraint reduces search to a single line, the epipolar line. How do you compute the epipolar line?

Given a point in one image, multiplying by the essential matrix will tell us the epipolar line in the second view.

$$Ex = l'$$

Epipolar line : $ax + by + c = 0$ $l = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$

If the point x is on the epipolar line l then $x^T l = 0$

$$\text{So: } x^T E x = 0$$

Reminder \rightarrow Vector product : takes two vectors and returns a vector perpendicular to both. Cross product of two vectors in the same direction is zero vector: $a \times a = 0$

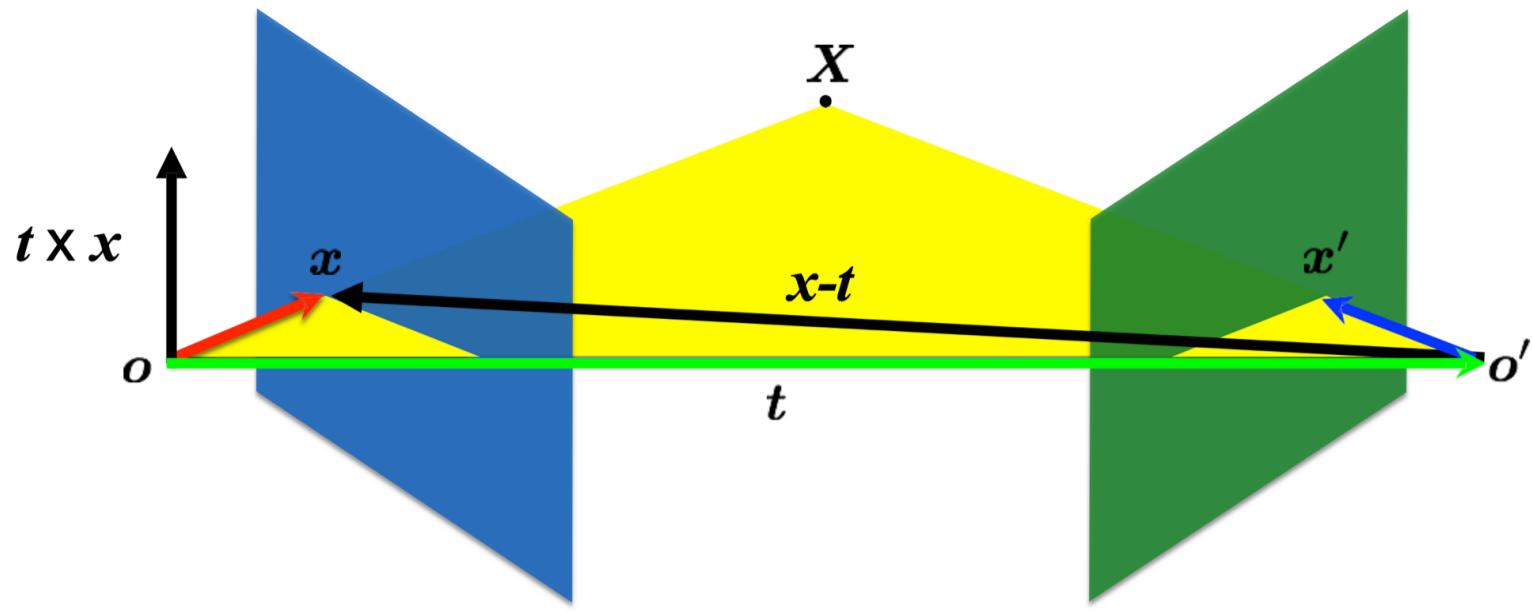
$$a \times b = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}^*$$

dot product of two orthogonal vector is scalar zero

Camera - Camera transform just like world - camera transform

$$x' = R(x - t)$$

R = rotation
t = translation



$$(x-t)^T \cdot (t \times x) = 0 \quad (\text{coplanarity})$$

dot product of orthogonal vectors

Combining the two forms:

$$(x'^T R)(t \times x) = 0$$

$$(x'^T R)([t]_x x) = 0$$

↳ Forma matriciale: Skew-symmetric matrix *

$$x'^T (R[t]_x) x = 0 \Rightarrow x'^T E x = 0$$

So:

$$E = R[t]_x \quad (\text{Matrix } 3 \times 3)$$

Knowing the Essential Matrix:

- $x'^T E x = 0$

- $l' = E x \quad l = E^T x'$

- $e'^T E = 0 \quad E e = 0$

Difference between the essential matrix and a homography:

- $U' = E x$: Essential matrix maps a point to a line
- $x' = H x$: Homography maps a point to a point

The Essential Matrix can be used under the Assumption:
2D points expressed in camera coordinate system (i.e., intrinsic matrices
are identities)

↳ Fundamental matrix is a generalization of the essential matrix,
where the assumption of Identity matrices is removed

From image point:

$$\hat{x}' = K'^{-1} x' \quad \hat{x} = K^{-1} x$$

Writing out the epipolar constraint in terms of image coordinates

$$x'^T (\underbrace{K'^{-T} E K^{-1}}_F) x = 0$$

F, Fundamental Matrix

Knowing F:

- $x'^T F x = 0$
- $U' = F x \quad U = F^T x'$
- $e'^T F = 0 \quad F e = 0$

Summary:

- Essential Matrix operates on points in camera coordinate system (after projection from 3D to 2D)
- Fundamental Matrix operates on points in pixel coordinate system
- E and F are both rank(2), but E has 2 singular values that are equal, but not F
- E has 5 DoF and F has 7 DoF (Degree of freedom)

8-point algorithm

The goal of the 8-point algorithm is to estimate the Fundamental Matrix F from at least 8 point correspondences between two images.

Given pairs of corresponding points (x_i, x'_i) (find with SIFT), the epipolar constraint states for each pair, the equation $x'^i F x_i = 0$ must hold. This constraint can be expanded into a linear equation in the unknown entries of F.

By stacking these equations for all point correspondences, we build a linear system of the form $A\bar{F} = 0$

$$\begin{bmatrix} x_1 x'_1 & x_1 y'_1 & x_1 & y_1 x'_1 & y_1 y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots \\ x_n x'_n & x_n y'_n & x_n & y_n x'_n & y_n y'_n & y_n & x'_n & y'_n & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_8 \end{bmatrix} = 0$$

To solve this linear system, we apply Singular Value Decomposition (SVD) to the matrix A . The solution f is given by the singular vector corresponding to the smallest singular value of A , which minimizes the algebraic error $\|Af\|$.

For better numerical stability, it is common to normalize the point coordinates before applying the algorithm. This normalization involves translating the centroid of the points to the origin and scaling them so that their average distance from the origin is $\sqrt{2}$.

Triangulation

Given : i) 2D point correspondences x_1 and x_2 in two images
 ii) the projection matrices P_1, P_2 for both cameras

We want to estimate the 3D point X

projection equation is : $x_i = P X$

$$X = \alpha P X \quad \text{similarity relation}$$

X and $\alpha P X$ same ray direction but differs by a scale factors

Cross product of two vector of same direction is zero

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \\ P_5 & P_6 & P_7 & P_8 \\ P_9 & P_{10} & P_{11} & P_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = d \begin{bmatrix} P_1^T \\ P_2^T \\ P_3^T \end{bmatrix} \begin{bmatrix} 1 \\ X \\ 1 \end{bmatrix} = d \begin{bmatrix} P_1^T X \\ P_2^T X \\ P_3^T X \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \times \begin{bmatrix} P_1^T X \\ P_2^T X \\ P_3^T X \end{bmatrix} = \begin{bmatrix} YP_3^T X - P_2^T X \\ P_1^T X - X P_3^T X \\ X P_2^T X - Y P_1^T X \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Third line is a linear combination of the first and second lines. We can remove it.

two rows
from camera one

$$\left\{ \begin{bmatrix} YP_3^T - P_2^T \\ P_1^T - X P_3^T \end{bmatrix} \times = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right.$$

two rows
from camera two

$$\left. \begin{bmatrix} Y'P_3^T - P_2^T \\ P_1^T - X' P_3^T \end{bmatrix} \times = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right.$$

$\underbrace{\quad}_{A}$

$A \times = 0$. We solve homogeneous linear system with SVD (Singular Value Decomposition)

Input	What You Can Estimate
2D-2D correspondences only (matches between two images)	Fundamental Matrix (F), Epipolar lines, Epipolar constraint (uncalibrated geometry), Intrinsic parameters via self-calibration (optional)
2D-2D correspondences + known intrinsic parameters (K, K')	Essential Matrix (E), Relative camera motion (R and t , up to scale), Normalized epipolar geometry
2D-2D correspondences + known camera poses (R, t) + known intrinsic parameters (K, K')	3D structure (triangulation), 3D point coordinates (X, Y, Z)
3D-2D correspondences (known 3D points + corresponding 2D projections)	Camera extrinsics (R and t), Intrinsic parameters, via calibration or bundle adjustment
Multiple images with no prior knowledge	Structure from Motion (SfM) or SLAM: 2D correspondences, Camera motion (R, t), 3D scene structure (point cloud), Intrinsic parameters (optional)