(2)

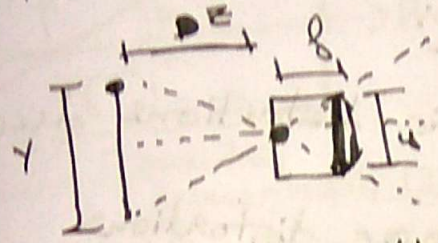
Human Eye:

- Retina → Fovea (High sensitivity)

——— x ———

* Pinhole Camera: (Everything is in focus)

- Focal length = f, Depth = z
- Inverted image formed.



$$\frac{Y}{z} = \frac{y}{f} \quad , \quad y = f \cdot \frac{Y}{z}$$

⇒ Issue is that a very small amount of light enters through the pinhole. (Long exposure)

- If pinhole size is large, we get blurry images since light from multiple regions are summed up in each point within the pinhole camera.

(Too small pinhole → Diffraction issues)

——— x ———

* Camera with Lens: (Bidirectional inversion)

* - Lens helps focus light in a range onto a single point. The aperture is large and yet we get a sharp image.
- $\frac{1}{f} = \frac{1}{d_o} + \frac{1}{d_i}$, f = Focal length of the lens.

To get a sharp image. d_o = Object to lens.
d_i = Lens to screen.

⇒ Depth of Focus (DOF) = Region of depth of object which appears to be in focus keeping camera at the same point.
(Depth of Field)

——— x ———

Note: Lens-Aperture Camera: (SLR cameras)

- Lens is present along with a variable hole size pin hole behind it.

Note: Zooming → f of lens changes or multiple lenses.
Focusing → Moving position of lens.

—x—

*Distortions due to lenses:

1) Pincushion & Barrel geometric distortions due to bad lenses.
ex: Fish eye lenses etc have these distortions

2) Lens Flare

- When there are multiple lenses, some light gets reflected back and light bounces between lenses / inside lens if its a single lens.

3) Chromatic Aberration:

- The focal point of different light R, G, B is not the same.

⇒ White light falling on lens, R, G, B gets split and we can see the light seperating out.

—x—

Note: CCD & CMOS sensor arrays for capturing light.

Note: Sampling: $f[m,n] = f(x,y) \sum_{n,m} \delta(x - nM_1, y - mM_2)$

* ⇒ M_1, M_2 = Sampling distances
⇒ δ's all together called 2D impulse train.

- We then perform quantization on $f[m,n]$.

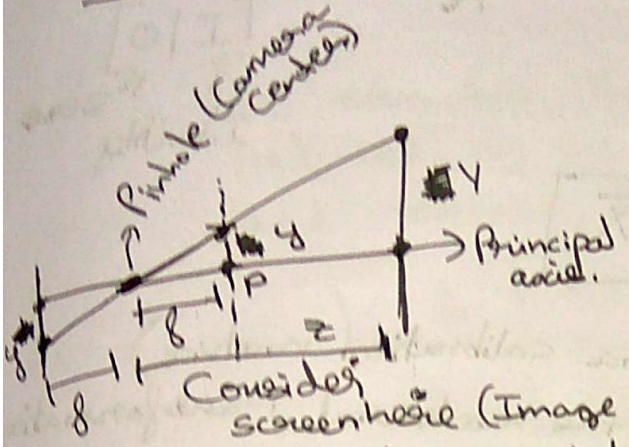
—x—

Note: Test on thursday (12th), 2.1–2.4, 3.1, 3.2.1, 3.2.2

~~(H&Z)~~

(3)

Geometry & Camera Matrices:



⇒ Consider screen ahead of pinhole for mathematical simplicity

(P = Principal point)

- In 2D, $x = \frac{fX}{z}$, $y = \frac{fY}{z}$

⇒ (x, y) are the coordinates on screen.
(X, Y, Z) are world points.

- $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ? \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$

- We cannot form this camera matrix without X, Y, Z in it, so we move to a different formulation

———x———

Homogeneous Coordinates:

- $\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$

$\bar{x}$ ↑ Camera matrix P $\bar{X}_c$

(Unit of f gives unit of x, y) (ex: Pixels)

, $\bar{x} = P \cdot \bar{X}_c$

- We interpret coordinates as $\left(\frac{x}{w}, \frac{y}{w}\right)$

⇒ Helps us model the nonlinearity with ease.

(Note: X, Y, Z are world point coords wrt coordinate axis of camera. (Camera coordinates))

$$\Rightarrow P = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}_{3\times4} = \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{K}{}_{3\times3} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}_{3\times4}$$

$$\therefore P = K \cdot [I|0]$$

$$\therefore \boxed{\bar{x} = K \cdot [I|0] \cdot \bar{X}_c}$$

$[I|0]$ → Identity, ← zeros

* (K = Internal camera calibration matrix)
([I|0] is what the external transformations affect)

———x———

Note: If pixel distances on screen are not same, change f's in K. (f's units are in Pixels), scale each f based on the pixel pitch of the corresponding axis. (pixel distances) (Internal property) (α_x β_y)

———x———

* Internal Properties:

a) We however want our image centre (0,0) to be at the bottom left corner & all coordinates $\bar{x}$ are +ve.

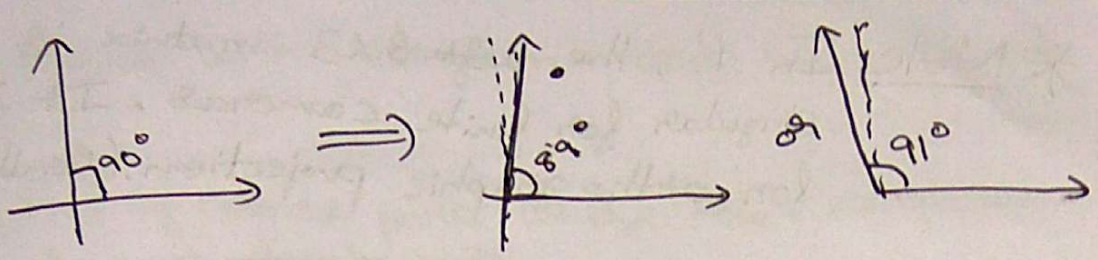
→ We do this by making

$$K = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$(x_0, y_0) = (\text{Height}/2, \text{Width}/2)$ of screen
(This is again internal to the camera)

———x———

b) Some cameras have issues where axes are not at a 90°. There is a skew. So to correct for the skew, x coordinate needs to be adjust



- We can see that x coordinate gets reduced, so we would have to modify and increase it based on skew (Y coord)

$$\therefore K = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

(Assume x axis is correct (convention))

$\therefore$ K = Upper triangular matrix with 5 degrees of freedom.

---

* External Transformations:

- Translation, rotation etc is done by using another matrix as required.

$$\Rightarrow \bar{X}_c = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix}_{4\times4} \bar{X}_w$$

, To get camera coordinates, we use another transformation matrix.

R = 3X3 rotation matrix

-RC = 3X1 vector

(Translation & then rotation) → Graphics notes for details.

Finally,

$$\Rightarrow \bar{x} = K[I|0]\bar{X}_c = K[R|-RC]\bar{X}_w$$

$$\Rightarrow \bar{x} = P \cdot \bar{X}_w = \underbrace{[KR|-KRC]}_{\circ} \bar{X}_w = [M|P_4] \cdot \bar{X}_w$$

→ Just a representation

Note:

- A few representations: $[P_1\ P_2\ P_3]^T$ or $\begin{cases} M = [P_1|P_2|P_3] \\ P_i = \text{Column vectors} \end{cases}$ →

where P_i = 4X1 column

So it becomes 3X4 matrix.

---

* Note: In P, the left 3×3 matrix is non singular for finite cameras. It is singular for orthographic projection. (Parallel projection)

—— x ——

(4) Camera Calibration: (Ch 2.1 to 2.3)

—— x ——

- Camera center does not have image on screen.

* (Camera center = World point that gives (0,0,0) when projected.)

- $P = [p_1\ p_2\ p_3\ p_4]$

* ⇒ Columns p_1, p_2 & p_3 are the screen coords of X, Y, Z vanishing points in the world.

ex: $[p_1\ p_2\ p_3\ p_4]\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ → X vanishing point → p_1 → Screen coords.

⇒ Column p_4 is the image of world origin.

⇒ $P = [p_1\ p_2\ p_3]^T$,

* ↳ p_3 = Principal plane = Parallel plane to screen passing through camera origin. (z=0 always)

$(a,b,c,d) \Rightarrow (a,b,c,0)$ ∵ Passes through origin.

⇒ (a,b,c) ⇒ Normal of plane/screen.

—— x ——

Camera Calibration: (Ch 2.1, 2.2, 2.3)

1) 30 Reference object based calibration ($\bar{x} = P\bar{X}_w$) (Use known $\bar{x}$, $\bar{X}_w$) (12 unknowns)

2) Precisely moving plane (Physical measurements)

3) Plane with unknown motion (checkerboard pattern on plane)

4) Set of collinear points where the line passes through fixed points.

5) Self calibration (rigid static world & point correspondences)

⇒ Going from ① to ⑤ precision drops, but easy to perform.

— x —

1) 3D reference object based: (Linear equations)

$$x_i = P_{11}X_i + P_{12}Y_i + P_{13}Z_i + P_{14}$$
$$y_i = P_{21}X_i + P_{22}Y_i + P_{23}Z_i + P_{24}$$
$$w_i = P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}$$

⇒ We know (X_i, Y_i, Z_i) & $\left(\frac{x_i}{w_i}, \frac{y_i}{w_i}\right)$ for each pair of world/screen points.

$$\frac{x_i}{w_i} = u_i = \frac{P_1\bar{X}_i + P_{14}}{P_3\bar{X}_i + P_{34}}$$

$$\frac{y_i}{w_i} = v_i = \frac{P_2\bar{X}_i + P_{24}}{P_3\bar{X}_i + P_{34}}$$

$$P_1\bar{X}_i + P_{14} - u_iP_3\bar{X}_i - u_iP_{34} = 0 \rightarrow (1)$$
$$P_2\bar{X}_i + P_{24} - v_iP_3\bar{X}_i - v_iP_{34} = 0 \rightarrow (2)$$

⇒ Using this we get P camera matrix. (once we find approx P)

⇒ We now need to decompose this

$$P = K[R|t], \quad K = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$ → This scales images ↓ Always an issue (cannot be found)

$$P = [M\ p_4], \quad M = KR, \quad p_4 = Kt$$

⇒ We know M & p_4, we need to find K

• In rotation R^T ⇒ Opposite rotation,

$\therefore RR^T = I$

Now,

⇒ $MM^T = KRR^TK^T = KK^T$. We can get K from this.

• We then divide K to make bottom right element 1 since that scales images.

• We can define ①, ② for each point in a matrix where $G_i \cdot p = 0$ (where $|p| = 12$ elements)

• G_i = Rows for each point

G_i = shape 2n×12

2n rows for n points

From this get R & t.

Make a thread G_i

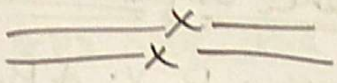
⇒ Refining P: (Non linear optimizations)

* • The P we calculate is an approximation, so we minimize error:

$$\min_{P} \sum_{i} \|m_i - \phi(P, M_i)\|^2$$

m_i: original position on image (u_i, v_i); $\phi(P, M_i)$: Points projected by P

⇒ Performs optimizations like gradient descent. (sort of) (Useful to remove noise etc that might be present in the image)

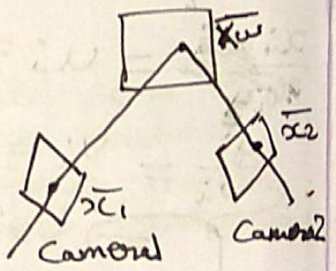


(6)

* 2-View Geometry: (2D World)

* • Assume world is planar & the XY plane. We can transform world plane to XY if its not originally on the XY plane.

$$\therefore \text{Coordinates} = \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$



$$\bar{x} = [P_1\ P_2\ P_3\ P_4] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

$$\bar{x} = [P_1\ P_2\ P_4]_{3\times3} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

↓ Non singular (Rank 3) matrix

$$\bar{x}_1 = H'_{3\times3} \cdot \bar{X}'_w = H_1 \cdot \bar{X}_w$$

$$\bar{x}_2 = H''_{3\times3} \cdot \bar{X}''_w = H_2 \cdot \bar{X}_w$$

$$\bar{x}_1 = H_1 \cdot \bar{X}_w = H_1 \cdot H_2^{-1} \bar{x}_2$$

$$\therefore \bar{x}_1 = H_{12} \bar{x}_2 \quad (H_{12} = H_1 \cdot H_2^{-1})$$

$$\bar{x}_2 = H_{21} \bar{x}_1 \quad (H_{12} = H_{21}^{-1})$$

- If we know H and one of the images, we can compute the other image automatically (Similar to interpolation on input for each output pixel)

*(H = Homography matrix)

⇒ H is a 3×3 matrix with 8 free variables. So to estimate it we need 4 point correspondences.

———x———

*2-View: (Same Camera Center) (3D World)

(K can change so zooming etc is allowed)

$$\bar{x}_1 = K_1 R_1 [I \mid -C] \bar{X}_w$$

$$\bar{x}_2 = K_2 R_2 [I \mid -C] \bar{X}_w$$

- Assumption is that camera center does not change. We only rotate it

⇒ K_1R_1 & K_2R_2 are full rank matrices

$$\therefore \bar{x}_2 = K_2 R_2 (K_1 R_1)^{-1} \bar{x}_1$$

$$\bar{x}_2 = H_{21} \bar{x}_1$$

↑ 3×3 Homography matrix.

⇒ Similar to previous case, there we assume world is planar but camera center can change. Here world can be 3D but camera center should not change

———x———

Note: other homographies are also possible, parallel lines etc.

———x———

Note: Panorama — Stitch images with homography Use merging techniques etc

———x———

*Feature Matching/Description/Detection:

- Feature detection, description & matching.
- SIFT, Harris corners, SURF etc

—— x ——

(7) Harris Corner Detection: (Feature Detection) (DIP Notes)

- Edges → Illumination edges, surface color edges
  → Depth edges
  → Surface normal edges. (example in slides)
- Depth & illumination edges depend on illumination of the scene & viewpoint.
- DoG → Difference of Gaussians operator.

$$D(G_\sigma(I)) \Rightarrow (D(G_\sigma)I) \quad (D \rightarrow \text{Derivative})$$

$$D(G_\sigma)I \rightarrow DoG(I) \rightarrow \text{Edge detection}$$

- $E(x,y) = \sum_{w\ \text{window}} (I(x+u, y+v) - I(x,y))^2$

$$I(x+u, y+v) = I(x,y) + \frac{\partial I(x,y)}{\partial x}\cdot u + \frac{\partial I(x,y)}{\partial y}\cdot v$$

$$\Rightarrow I(x+u, y+v) - I(x,y) = \begin{bmatrix} I_x^{(x,y)} & I_y^{(x,y)} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\Rightarrow E(x,y) = \sum_w \left[\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \right]^2$$

$$= \sum_w \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \sum_w \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}}_{H\ \text{matrix}} \begin{bmatrix} u \\ v \end{bmatrix}$$

(As we vary x,y → We want to identify minimum of $E(x,y)$ & then maximize that → for all (u,v)

i.e. Maximize the direction (u,v) giving min change → min eigenvalue

⇒ Eigen vectors & values for H give us the direction & amount of change in those two major directions.

⇒ $Hx_+ = \lambda_+ x_+$, $Hx_- = \lambda_- x_-$

λ_+: value; x_+: Max change vector

λ_-: value; x_-: Min change vector

- We want min change vector value $= \lambda_-$ to be high. ie. Both eigen values should be large. ($\because \lambda_- < \lambda_+$) (easier to check)

⇒ λ_+ = High for edges, λ_- is also high for corners.

- $f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{\text{Determinant}(H)}{\text{Trace}(H)}$ → Harris operator

If f is large then select point.

⇒ We then apply local thresholding to select single points in a region so that we pick only the local maxima.

(8)

———x———

Feature Description:

*1) Multiscale Oriented Patches: (MOPS)

- Take multiple windows of various sizes around feature points & store them as the descriptors.
- For orientation of windows, use the direction of maximum gradient as the base orientation
- 8×8 patches at 5 scales (scale image)
- Bias/gain normalization to remove mean & divide by variance → illumination invariance.

———x———

2) Scale Invariant Feature Transform (SIFT)

- Illumination Invariance — Difference based metrics (Difference between points etc)

*- Scale Invariance — Pyramids (resized images)
                    — Difference of gaussians (Subtract levels from gaussian pyramid without resize)
                        ↑ Different blurs.

(Blurring is sort of like continuous scaling) → In DOGs, after certain amount of blur, we lose as much info as we lose on resize, so to save space we resize the image & then proceed from there.

- Rotation Invariance → Histogram of gradient directions for all pixels in window. Base reference direction is average direction of all gradients.

(These are stable to illumination changes)

⇒ Create 4X4 regions in window & get histograms of orientations in each region, the gradients are given weighted contributions.

- Gradients of pixels are slightly averaged, so that different region spanning edges are taken care of.

⇒ Feature detection → Max in multiple levels of difference of gaussians (DOGs)

- Feature size: 16 X 8 = 128 dimensions.
  ↑ Regions ↑ Histogram with 8 elements.

———— x ————

* Feature Matching:

- Direct matching → n^2 comparisions of features (If n points are present)
  (Too slow)

⇒ Hashing of descriptors → Matches are directly found
→ Hard to hash though since features are similar but not identical.

⇒ Nearest neighbours → k-d trees etc.
* (Approximate algorithms that are fast)

- Outliers can be dropped using thresholds on match similarities.

———— x ————

* RANSAC:

* • Select 4 feature pairs at random (from matches)
  • Compute homography
  • Compute inliers where $SSD(p_i', H p_i) < \varepsilon$
  • Repeat this process and the one where we get maximum inliers, keep them and discard outlier matches.
  • Finally compute H' on all inliers. This is the final homography matrix to align images based on features.

———— x ————

⑨ # Geometry (3D world + Camera center Shift allowed.)

- Consider 2 cameras, you know the rotation & translation between these 2 cameras

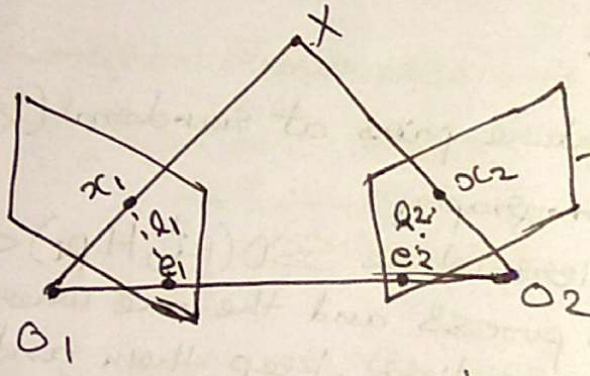
⇒ If a point X in 3D world appears at x_1 on camera C_1 with origin O_1 & X appears at x_2 on camera C_2 with origin O_2

* X lies on intersection of $\overline{O_1x_1}$ & $\overline{O_2x_2}$. (Essentially we are identifying the depth as well) → Reconstruction of 3D world from images. (Slight errors in x_1 & x_2 give huge errors) (Noise in image etc)

---

* Epipolar Geometry:

**



- e_1 & e_2 are epipoles → They dont change if we change X.

⇒ All world points mapping to x_1 in I_1 (pre image of x_1) map to a line l_2 in I_2. l_2 is essentially the line on the image plane of I_2. l_1 & l_2 are called epipolar lines.

- Image of O_1 in I_2 = e_2 & image of O_2 in I_1 = e_1 are epipoles.
- The plane of O_1e = Epipolar plane

Epipolar Constraint:

* • $\lambda_1 x_1 = X$ } since x_1 & X lie on same line (Assume origin at O_1)

$\Rightarrow \lambda_2 x_2 = RX + T$, where R, T are rotation & translation of O_2 wrt O_1.

$= R(\lambda_1 x_1) + T$

- Additions are inconvenient, so we try get rid of it with $\hat{T}$

- $\hat{T}\lambda_2 x_2 = \hat{T}R\lambda_1 x_1 + 0$, $\hat{T} = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ T_y & T_x & 0 \end{bmatrix}$

$\Rightarrow \lambda_2 x_2^T \hat{T} x_2 = \lambda_1 x_2^T \hat{T} R x_1$

$A \times B = \hat{A}B$

- $x_2^T \hat{T} R x_1 = 0$

$\because x_2^T \cdot \hat{T} x_2 = 0$

Cross product, ⊥'lar to Δ'le plane (Dot product with 90° = 0) //

$\therefore \boxed{x_2^T E x_1 = 0}$ or $x_1^T E x_2 = 0$

↑ If we repeat the process with O_2 as origin.

we can find this as above

- Now $A \times A = 0$

~~[illegible]~~

$\because A \times A = \hat{A}A = 0$

- We know $E = \hat{T}R$ (Since we know how our cameras are configured)

$\Rightarrow$ Now if we know x_2, since we know E we can get $x_2^T E$ = 1×3 vector and,

$0 = (x_2^T E)x_1 \Rightarrow$ Epipolar line l_1 → x_1 lies on this line.

Similarly, $0 = (x_1^T E)x_2 \Rightarrow$ Epipolar line l_2 , E = Essential matrix *

Note: If internal parameters of C_1 & C_2 are different (K_1 & K_2) *

then, $x_1 = K_1 X$, $x_2 = K_2 X$

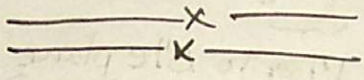
we get, $x_2^T \underbrace{K_2^{-T} \hat{T} R K_1^{-1}} x_1 = 0$

F → Fundamental matrix

Note: The reason we move to epipolar geometry * is because finding x_2 corresponding to x_1 we need not search the entire image but just on l_2.

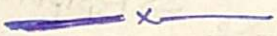
Note: * To find X, we need x_1 & x_2. To get corresponding match of x_1 using SIFT etc, we need to search entire image for feature matching but from epipolar constraint, given E & x_1 we only need to search on l_2 for our match point x_2. (Reduces search space).

- Once we have x_1 & x_2, we can find X as intersection of O_1x_1 & O_2x_2.



(10) Multiview Reconstruction:

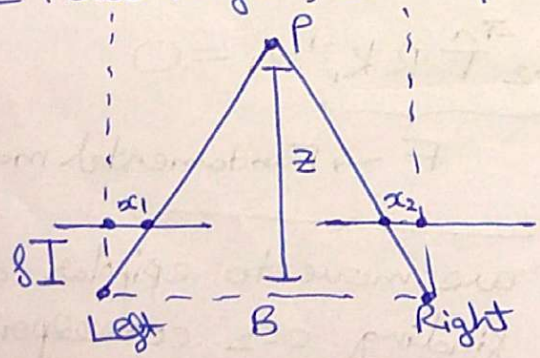
- Binocular stereo — Two cams with known intrinsic and extrinsic params
- Multiview stereo — Multiple known cameras
- * Structure from motion (SfM) — Multiple cameras & projections with correspondences. Retrieve 3D coords & camera matrices C_i.



* Binocular Stereo:

- Assume both cameras are similar (internal params)

⇒ f = Focal length, z = Depth



- $\frac{x_1 - x_2}{f} = \frac{B}{z}$ (-ve direction)

from similar triangles.

$\Rightarrow z = \frac{Bf}{x_1 - x_2}$ //

* $\rightarrow x_1 - x_2$ = Stereo disparity or Parallax. (Δ) $= \frac{Bf}{z} = \Delta$

$x_1 - x_2$: Same world point in both images.

↳ As $z \uparrow$, parallax $\downarrow$

→ Larger baseline gives better estimates but hard to get matches etc.

↳ Since we want to avoid Δ errors ($\Delta \to 0$)

Note: Matching patches (vectors)

* • Correlation b/w v & v' → $\frac{v'^T v}{\sqrt{v^T v} \cdot \sqrt{v'^T v'}}$

• Normalized correlation → Subtract mean from vectors & correlate.

Note: Constraints on matching:

* • Epipolar lines • Colour constancy

• Uniqueness — one point on one image only matches 1 other point. Not always true.

* • Ordering & Monotonicity: Points appearing in an order will remain same for small camera rotations or translations.

Note: In Binocular Stereo, if one of the cameras is rotated as well instead of just translation, (Rectification) ↓ ** image I_2 convert to I_2' such that I_2' is rotation of I_2 (Affine transform). Compute & apply the rotation homography & you can now perform binocular stereo.

Camera calibration for stereo ↓ Estimate H_1 & H_2 → 2 homographic transformations

(If translations along both axis b/w cameras, perform parallel to line joining their centres)

Note: Dense pixel-pixel matching — DP Algos. (Slides)

* (Not just matches b/w certain feature points)

↓ Globally optimal solution. → Some issues exist as well.

———x———

Structured Lighting:

- Project patterned light onto the world & then captures images so that features & match detection becomes simpler.

(Kinect uses this)

———x———

———x———

Note: Things trivial for humans are hard for computers → Grouping (segmentation)

Things trivial for computers are hard for humans → Measurements

Image Segmentation:

- Classification based → Each pixel is classified
- Region based → Region growing & splitting
- Boundary based → Edges
- Motion based

* Thresholding: (DIP Notes)

- Global → Threshold selected via histogram.
- Adaptive (Local) →

1) Global → Automatic Thresholding

- Select threshold T
- Compute mean values μ_1 & μ_2 of each segment
- $T = (\mu_1 + \mu_2)/2$, repeat until convergence.

→ Multispectral automatic thresholding

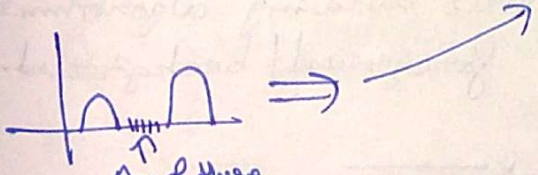
- Perform automatic thresholding on each channel. Use binary of classes as new class.
  ex: 01, 10, 11, 00 } 4 classes (2 spectral)

→ Multithreshold automatic

- Compute multiple thresholds automatically.

→ Otsu's method

- Bruteforce over thresholds
- Choose one which maximizes between class variance. (squared distance blw means)



Any of these give same results pick midpoint

~~Adaptive thresholding~~

2) Adaptive Thresholding

- Use mean in a local region. (Generally bad)
- * Mean-C
  - Uniform region set all as background otherwise perform mean thresholding as above.
- Chow-Kaneko
  - * Apply mean filters multiple times to get background image and subtract from original.
  - Then apply Otsu etc

Note: Optimal Thresholding: Fit multiple gaussians to replicate given histogram as closely as possible.

* (Gaussian Mixtures) (EM Algorithms) (SMAI Notes)

(Intersection of gaussians is the threshold) (Global thresholding)

Graph Cut based Binary Segmentation

- Interactive method where foreground & background are marked with lines allowing algorithm to create a model of foreground/background.

MRF

Superpixels:

- If adjacent pixels are quite similar, group them together to form a superpixel.

⇒ Essentially these superpixels are highly grainy segmentations.

* Graph Cuts: (Normalized Cuts paper)

* - Creating graphs directly where each pixel is a node makes them hard to handle, so divide the image into superpixels and then perform graph operations.

⇒ Adjacent superpixels are connected by edges with their weights based on similarities.

⇒ Perform min cut where we can then split the foreground and the background using this cut.

- $Cut(A,B) = \sum_{u \in A, v \in B} w(u,v)$ } Performing direct min cut usually just puts one superpixel in a region & seperates the rest.

∴ We use normalized cuts.

* $NCut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{Cut(A,B)}{assoc(B,V)}$

↓ Minimize this

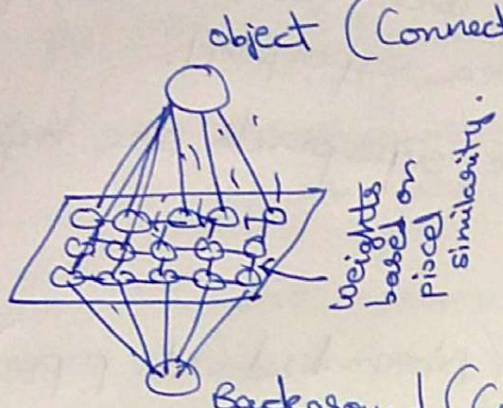
↙ Total Number of edges going out from all elements in A + edges in A.

(Read the paper)

———x———

* Binary Image Segmentation: (Graph Cut Based)

* • Perform graph cut but add model information as well.



⇒ Ask user to specify 2 lines, one on the foreground and one on the background

- Strength of connection from Object to pixel & Background to pixel depends on the input user gave (essentially we are adding in class information)

⇒ Graph cut where one constraint is added that "Object" & "Background" should belong to different sides of cut

$E(x) = \sum_i c_i x_i + \sum_{ij} c_{ij} x_i (1 - x_j)$

———— x ————

Note: Energy function: (examples in slides)

- Assign labels to vertices such that overall cost is minimized. → Energy function (Overall cost)
  (Similarity: Potentials)
- We want to minimize the energy
- If vertices are connected, define additional cost as. $\alpha(a,b) = w(f(a) - f(b))$
  - $\alpha(a,b)$: some cost between edge ab.
  - w: weight of edge

where $q(a,a) = 0$, $q(a,b) = 1$ (Same labels adjacent pixels add no cost whereas different labels add cost)

↑ You can define any costs here.

$\mathcal{O}$ = Pairwise potential

* ∴ Finally $Q(f) = \sum_a \mathcal{O}_{a:f(a)} + \sum_{(a,b)} \mathcal{O}_{ab:f(a)f(b)}$

- $Q(f)$: Cost of a certain labelling
- $\sum_a \mathcal{O}_{a:f(a)}$: Unary potential (Cost of labelling)
- $\sum_{(a,b)} \mathcal{O}_{ab:f(a)f(b)}$: Pairwise potential (Cost due to connected vertices)

$\Rightarrow q^* = \min Q(f;\mathcal{O}) = Q(f^*;\mathcal{O})$

- This is a huge problem: 2^{vertices} assignments possible.

$\Rightarrow$ Cost of a source/sink cut = Sum of edges on cut going from source to sink.

- these energy functions are minimized via mincuts.

---

*Note: Max flow algorithms:

- Start with flow 0
- Find path with true flow from source to sink and push max into it as possible and add to flow.
- Subtract ~~these~~ capacities from this pushed cap and add to back edges.
- Repeat.

$\Rightarrow$ Mincut = Edges whose capacity is 0 after max flow. Value of mincut = max flow obtained.

Note: Papers on energy minimization. Fast approximation etc.

Note: Maximum Apriori P?

(1) Grab Cut : (Segmentation)

* • Specify region of image that does not contain the background. (To create a background/foreground model).

⇒ Uses graph cut for segmentation.

⇒ Foreground/Background models based on mixture of gaussians.

• Iterative graph cut procedure where we start with region outside rectangle as background & inside rectangle as foreground. → Perform graph cut & then based on assignment, retrain foreground/background models & repeat the process till convergence.

⇒ Foreground/Background border is separated via matting (smooth transition) (Like a sigmoid)

———— x ————

(2) MRFs for Binarizing Natural Scene Text

* • Text Detection (Region selection) → Binarization → Character segmentation → Recognition

⇒ Pairwise & individual energy terms.

⇒ Also added an ~~edginess~~ edginess term to the energy → Generally on segmentation boundary there are edges.

⇒ Heuristic to create foreground / background models → Perform edge detection & if edge near top, above it is background. Two parallel nearby lines → area b/w them belong to the foreground.

→ Perform standard graphcut.

———x———

Note: OBJ CUT paper. Read GrabCut & MRF ② papers.

———x———

→ Graph Based methods

Multilabel MRF (Semantic Segmentation)

———x———

• More than 2 labels makes graph cut NP hard.

↳ We use α-expansion approximation algorithms

↳ Approximation quality = 2 x Best. (Slides)

(Potts model is an example)

———

* Metric Functions:

• $V(\alpha,\beta) = 0 \iff \alpha = \beta$ } Semimetric functions.

$V(\alpha,\beta) = V(\beta,\alpha)$

$V(\alpha,\beta) \leq V(\alpha,\gamma) + V(\gamma,\beta)$

$$\Rightarrow E(f) = \sum_{\{p,q\} \in N} V_{p,q}(f_p, f_q) + \sum_{p \in P} D_p(f_p)$$

↑ Energy function generalized.

———

Note: Move based algorithms: Have a local search neighborhood and move to best solution in that region.

↓

Keep moving as long as you are improving on your solution.

* α-Expansion Algorithm: (Semantic Segmentation)

1) • Start with random label assignments to pixels.

* • Move over each label α performing binary classification (α or Rest) and then check if score increases. (we only move pixels not having α to existing labels, hence the name α-Expansion).

⇒ If your energy function does not reduce any further, then stop. (Expansion does not change assignment anymore)

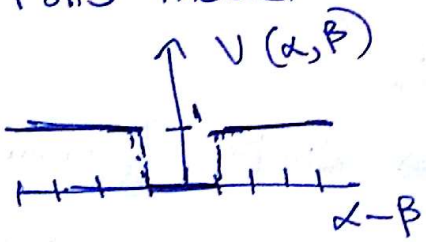
(Example in slides)

(Local algorithm (might get stuck at local optima at times))

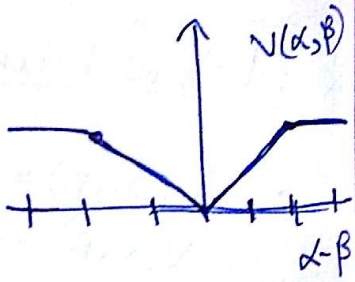
• At each step we perform graph cut since its only 2 classes now (α & $\sim\alpha$)

2: Note: Potts model

↓ Read up.



Truncated linear V



• These models are used in energy functions which are used on graphs.
(Create graph cut & now check if the energy function is minimized after cut)

• This linearity helps Counter structured noise in images while classification etc.

---

*2) Ink Bleed: (Removal)

* • Ink written on other side of paper bleeds through.
* ⇒ Front (Recto) & Back (Verso) sides of the paper.
  - Flip the Back and align the two images.
  - We assume we have 3 labels - Foreground, Background and Bleed.

⇒ Dual Layer MRF: First layer has vertices from front image
Second layer has vertices from second image.

- We define weights between each node in these layers (corresponding) such that the cost function is defined such that "Ink Bleed – Ink Bleed", "Ink Bleed – Background", "Background – Ink Bleed" etc get a large? value since we don't want them (we know it's not possible)

⇒ Unary assignments:

$$E(l = F) = \frac{S_I + S_B}{2(S_F + S_I + S_B)}$$ (Foreground) → From model (Similarities) (SVM outputs)

$$E(l = B) = \frac{S_F + S_I}{2(S_F + S_I + S_B)}$$ (Background)

$$E(l = I) = \frac{S_F + S_B}{2(S_F + S_I + S_B)}$$ (Ink bleed)

⇒ Pairwise: $E_S = \sum_{p,q \in N} V_1(l_p, l_q) + \sum_{p,p' \in M} V_2(l_p, l_{p'})$

Matrices define these costs

V_1: Intra layers (different class assignments get a large? cost)

V_2: Between Layers (we give preference to a few pairs) ($S_1 = S_2$)

- Then use α-expansion to solve the problem.

Note: Ink Bleed removal instead of segmentation (To retain the background as much as possible)

* → Foreground
→ Background
→ Bleed 0.05
→ Bleed 0.1
.
.
.
→ Bleed 1.0

} Labels

- $Ink_p = I^{(\alpha_p + \beta_p)} \cdot Background_p$ } Light absorption physics

  ↓ Estimate α_p & β_p for all pixels

  ↓ Ink ↓ Bleed.

- Then perform standard α-expansion after defining the energy functions.

———— x ————

Note: Depth/Disparity estimation, Normal/Shape estimation, Image restoration/enhancement.

———— x ————

* Face Detection with Eigenfaces:

- Face Image → Vector (n x m) → Perform PCA using training data

  ↕

  Reproject with minimal loss → If error is minimal than its a face vector. → Perform classification (Recognition)

  For classification → Determine other stuff using this as feature vectors.

———— x ————

Face Detection: (Viola and Jones)

* Try every window & check if its a face or not. This is too slow since there are too many windows.

→ False positive rate should be very low and detection rate should be high.

i) Image Features:

* Use binary masks (+1 & -1) convolving with the image where feature values are the convolution sum. (Bright − Dark)

⇒ Some features give true +ve values at certain parts of the image which we can use to figure out parts of the face.

* To compute these binary convolutions, we need rectangular sums of the image, this is done very quickly using the integral image.

ii) Window rejection: (Speedup) (Cascading)

* Prioritize binary masks so than if some mask fails at a window then we can directly reject else we try more masks and check if the features are valid or not (to confirm faces)

Perform at multiple scales (Image pyramid) for scale invariance (Windows used)

iii) Boosting: (Adaboost)

* Weak learners → features that perform just a little better than random.

↓

Each filter that is thresholded → Adjust the threshold so that no face is missed (spurious are allowed)

- We then combine multiple of these using Adaboost → to create strong classifier

⇒ Since our thresholds are adjusted such that even a single reject means it is not a face. We have fast window rejection here.

Note: Cascading is using a series of classifiers. accept = move to next classifier, reject = stop. As we move forward classifiers are made more complex using multiple weak learners at once using Adaboost.

Note: Training is very slow, to create classifiers using Adaboost requires multiple rounds etc. Testing is quite fast though.

(Boosting illustration in slides)
(Cascade structure in slides)

Pedestrian Detection
(HoG + SVM)

↳ Generative / Discriminative models

↳ Global / Part based descriptors

↓ Handles low res images well.

Each part of the human body
↳ Handles occlusions etc well

- Gradient Histograms (SIFT, GLOH, HOG etc)

→ Gradients: [-1|+1] } One sided
[-1|0|+1] } Two sided
} Then magnitude & angles

- Training data → Crop people → ~4000 dimensional HoG features

Binary classifier (SVM) ←

i) ⇒ Detection → Use scale space of images and combine adjacent window detections (sliding window)

↓
64X128 pixel windows

ii) Descriptor → • Compute gradients. (Two sided derivative)

* • 8X8 pixels treated as cells and compute histogram for each cell giving 8X16 cells.

• Use a 2X2 ~~overlap~~ cells as a block and normalize the histogram. These blocks have a 50% overlap as well. This gives us 7X15 blocks in total.

← Different kinds of normalization possible.

L2 was good.

(For each pixel, gradient = max gradient over all channel

• Gradient histogram uses 9 bins.

* • Trilinear interpolation of gradients

* → Gradient magnitude linearly interpolated between nearest 2 bins based on angle.
  → Bilinear interpolation of pixel gradient in the closest 4 cells.

(Overall, the magnitude is split twice, $\frac{m}{x_1 - x_2}$, both these interpolations together is called trilinear interpolation)

Note: Gradient image is calculated only once initially to speed up the algorithm.

* ⇒ Total feature shape = $7 \times 15 \times 4 \times 9 \approx 4k$.
  (dimensions) — Blocks, Cells in a block, Bins in a cell.

Note: Support Vector Machine Detector, Dynamic Pedestrian detection (Viola & Jones extension), similar feature detections using binary filters HoG however beat them all. Extension of HoG pedestrian → 2d Global Detector paper.

---

* SVMs:

- $R(\alpha) \leq R_{train}(\alpha) + \sqrt{g(h)/N}$
  ↑ Risk during testing assuming data distribution is the same.

- h = VC dimensions, g(h) = Monotonically increasing function.
  ∵ We want h to be small to get better test results.

⇒ $h \propto \frac{1}{\text{Margin}}$ → of the SVM

• This margin is relative to the diameter of hypersphere over all the points.

⇒ Relative margin: ℓ / D

$$h \leq \min\left\{d, \left[\frac{D^2}{\ell^2}\right]\right\} + 1 \quad \} \text{ VC Dimension}$$

(d: Dimensionality of samples.)

⇒ This minimization statement disproved the curse of dimensionality, we just need h to be small for low errors.

∴ We formulate an optimization problem to maximize the relative margin. (SMAI Notes)

• Only support vectors influence the boundary (w)
($wx + b$ = linear classifier)

i) Allow slack variables for noisy data

ii) Kernels for non linear boundaries
→ $K(x,y) = (1 + x \cdot y)^n$ } Huge dimension expansion
x

(Note: Mercer's Theorem → Rules for kernel functions)

x

*Visual Bag of Words:

- Histogram of words → Document summary (Use dictionary)
  - ↳ Spatial info is lost
  - ↳ Syntax/Grammar is lost

- Bag of features models for images
  - ↓ visual words

---

i) Texture Recognition:

- Repetition of textons (units of textures)
- Use bag of words → Histogram of all possible textons (dictionary)

(Comparing histograms with distance measures)

ii) Bag of Words Model:

- Histogram of words (Tag clouds etc)

I) ⟹ **Feature extraction:** Compute visual features from all images to learn a visual vocabulary
  - ↳ Quantize to get a finite vocab.

** ↳ 1) **Patch retrieval:**
  - i) Regular x grid of a specific size.
  - ii) Interest point detection & patches around them. (Ellipses around them etc)

2) **Features:**
  - Normalize patch & extract SIFT feature

II) Learning visual vocab:

* • If SIFT features of some patches are too close to each other, we should consider them as only one word in our vocab.
  - Apply a clustering algorithm which would give you a vocab of best (most dissimilar) features. (cluster centers) (K-Means clustering)

→ This vocab is our "codebook".

III) Histogram creation: (for test data)

- Given an input image, extract features in a similar manner and then hard cluster it into one of the elements of our codebook (nearest neighbour)
- Now use the histogram generated for other tasks.

Note: Scale issues in images → Pyramids or divide image into regions & get individual histograms.

Note: Dictionary size? Computation efficiency → Vocab trees paper ?
Image retrieval using histograms. Generative or discriminative models while training - Sparse visual histograms usually → can be used for speeding up matching etc:

* Uses: (Visual histogram (bag) of words)

i) → Image classification: Discriminative methods, standard classification techniques

ii) Topic discovery: Learn about the space from which data is drawn

iii) Action recognition: dictionaries specific to task at hand.

iv) Image search:

↳ Copyright issues → discover these
↳ Retrieve similar images etc.

———x———

Note: Pascal VOC } original object detection & localization challenge

Note: Video Google (paper) → Features from videos, similar videos, video search etc.

———x———

Feature Learning with DL:

- Perceptron learning
- Multi layer perceptron → First autonomous vehicles
- CNNs — Animal neuron receptive fields gave idea.
- Dropout

$$\Rightarrow \frac{(N-F)}{S} + 1$$

N → Inp size, F → Filter size, S → Stride

} Conv layer output size ($N' = N + Pad$)

- Max Pooling → Useful (Preserve maximum activations)

*(AlexNet, ZFNet, VGGNet, GoogLeNet (Inception modules), ResNet (152 layers))

Image Indexing & Retrieval:

- Content based image retrieval (CBIR)
  - ↳ Attribute based
  - ↳ Object based
- CBIR → Database population phase
  - ↳ Retrieval phase

Note: "Things" → Foreground important objects
"Stuff" → Background objects.

⇒ 1) Feature selection for Database population:

- Initially, color histograms, color moments, texture descriptors are used, for retrieval, use a distance metric for similarity.
- Practical issues with huge amounts of data & specific subdomains with specific matchers — faces, monuments etc.

- Text indexing & retrieval is well explored. converting images into text documents can help perform this well.

i) Virtual Textual Representation:

- Get segments of image that seem interesting
- Transform segments into text.
  - ↳ Then perform quantization to bin these segments. (Each cell has a representative string, so each segment now has a text string)

2) Inverted Index:

- For every word in an extended dictionary (all names, etc) have an index which points each word to all the documents and where it occurs. (Crawling to build the index)
- Based on this inverted index, given a query, we can try find the most relevent documents.

⇒ Documents can be represented as word histograms.

- Inverted indexes still however give you a lot of results and then page ranking needs to be done to get the best results. Page ranking generally depends on number of other sites linking to the page.
- Weighted word histograms.

* 3) Document Representations:

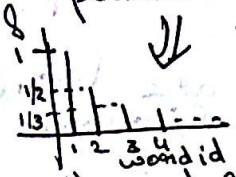
i) Binary weights: * $A_{ij} = 1$ if document i has word j or else its 0.

ii) Raw term frequency

** iii) TFIDF:

- Get frequency of each word over all documents you have → Document frequency
- Get frequency of words in relevant documents. → Term frequency

It was noted that ratio of these frequencies in decreasing order had a specific pattern:



Zipf distribution

- i^{th} word frequency $\propto 1/i$

⇒ Now if TF is high & IDF is low then we consider the word as important.

- ∴ TF X IDF → weight for each word
  - IDF → Inverse Document frequency.

* $w_{ik} = tf_{ik} \times \log(N/n_k)$ ← idf_k

- w_{ik} → weight of term T_k in document D_i
- tf_{ik} → Frequency of term T_k in document D_i
- N → Total documents in collection
- n_k → Number of documents in collection containing T_k

- Take all documents selected in Inverted Index & then rank them based of TF with search terms. Larger documents have larger TFs, so we need to normalize this bias.

$$\therefore w_{ik} = \frac{tf_{ik} \cdot \log(N/n_i)}{\sqrt{\sum_{k=1}^{K} (tf_{ik})^2 (\log(N/n_k))^2}}$$ → Normalize

*Optical Flows & Tracking

- Feature tracking – sparse correlation
- Optical flow – dense correlation (every pixel's motion is calculated)

⇒ At time t, pixel at (x,y) moves to $(x+u, y+v)$ at time $t+1$. (u,v) = Shift of pixel at (x,y)

⇒ $I(x,y,t) = I(x+u, y+v, t+1)$

- Using taylor approx,

$$I(x+u, y+v, t+1) = I(x,y,t) + I_x \cdot u + I_y \cdot v + I_t$$

(I_x, I_y & I_t are derivatives of image.)

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

$$* \Rightarrow \boxed{\nabla I \cdot [u\ v]^T + I_t = 0} \rightarrow (1)$$

- This cannot be used since each pixel has only 1 equation but 2 variables u, v to solve.

⇒ We add in another constraint, if * we take a vector parallel to edge (perpendicular to gradient), $\nabla I [u'\ v']^T = 1$ then (1) should be satisfied by this as well & $(u+u', v+v')$ satisfies it.

Slides ↳ (The Aperture Problem demonstrates this) (Motion perpendicular to gradient is not visible)

Motion along direction of edge is not noticeable, so if actual direction $= (u,v)$ for a pixel, then if (u',v') = direction along edge then

$(u+u', v+v')$ direction is also a valid solution so it would satisfy equations ①.

Note: this can also be observed in audio.

---

i) Spatial coherence constraint: (Lucas-Kanade) (L-K)

* To solve 1 equation 2 variable issue, assume adjacent pixels move in the same direction.
* ex: 5×5 window to get 25 equations for 2 variables. (Overconstrained)

$$\begin{bmatrix} I_x(p_1) & I_y(p_2) \\ \vdots & \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}_{A} \begin{bmatrix} u \\ v \end{bmatrix}_{d} = - \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}_{b}$$

Get (u,v) using pseudo inverse.

* $Ad = b \Rightarrow (A^TA)d = A^Tb$

$$d = (A^TA)^{-1}A^Tb$$

$$A^TA = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

$$A^Tb = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

Essentially Det should not be 0,

* If plain image or just vertical/horizontal gradients would make det 0

(Eigen vectors of A^TA should both be large ↓ Similar to Harris conditions)

↙ This would ensure our predictions of (u,v) are correct and not ambiguous.

---

ii) ⇒ (Lucas Kanade – Iterative) (Algo in slides)

Large movements are hard to approximate directly with linear approximation, so we apply the same L-K procedure multiple times to get the total motion (Iterative updates)

(Spatial coherence constraint)

- Apply algo on image → (get (u,v)) → Apply shift to image

Repeat (Until changes are minimal)

iii) Use scale space pyramids to approximate larger changes well. → on spatial coherence constraint. (L-K Algo)

(Can be iterative as well) at each level

---

* KLT - Tracking:

- Similar to flow algorithms
  - ↳ But also uses second moment matrix for better approximation.
- Find good points to track — Harris corners.
- Iterate & coarse-to-fine search for large motion.
- Small windows — sensitive to noise, miss large motion without pyramid.
  - ↳ 15×15, 31×31 are common. (weighted windows used)

Motion & Perceptual Organization

↳ Grouping of objects based on motion.
(Even motion with multiple points helps us group better with a lot more confidence)

Applications of optical flow,

Note: Segmentation based on motion cues, learning and tracking dynamic models, activity recognition and improving video quality. MPEG-4 video compression.

* Motion Field :

- Ideal optical flow (Ground truth sort of)
- Compute actual motion on 3D objects & project onto the image. → Motion field.
- Optical flow we compute from just images is apparent motion. (Approximate)

⇒ We want optical flow to be equal to motion field.

(Optical flow is based on illumination changes, Motion field is based on actual motion of objects)

———— × ————

Note: Further approaches have added a lot more on top of this. Dense Flow → Read.

———————

Projective Geometry: (Recap)

- $\bar{l} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$, $\bar{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ → Homogeneous coords. in 2D
  → Ray in 3D through origin.
  ↗ Intersection of rays from (0,0,0) to (x,y,1) on plane z=1 (3D→2D)

$\bar{l}^T \cdot \bar{x} = 0$ } Equation of a line (All points on line if $\bar{l}$ = const)
↳ Set of lines through $\bar{x}$ if $\bar{x}$ = const.

- Points at infinity $\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$ can be seen as (∞, ∞) but in the direction (x, y). → Ideal point in direction (x, y)

⇒ $[x\ y\ 0]^T$ ↳ are all points at infinity, they * together form the line at infinity $[0\ 0\ k]^T$

- Lines $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ on projection plane can be thought of as a plane through
* origin intersecting the $z = k$ plane. (In 3D)
* $(aoc + by + cz = 0$ if (x, y, z) lie on line and (a, b, c) is ⊥'lar to the line, so if we think of it as a plane (a, b, c) represent the normal of the plane)

⇒ The line at infinity $(0, 0, k)$ is the plane that is ⊥'lar to our projection plane (since they intersect at ∞).

---

* Line joining 2 points:

- $l^T p = 0,\ l^T q = 0$

$$l = [(y_2 - y_1)\ \ -(x_2 - x_1)\ \ (x_2 y_1 - x_1 y_2)]^T$$

(In normal 2D lines)

⇒ Thinking of it in terms of plane through origin in 3D, each p, q are rays from origin and line is the plane containing these 2 rays. So normal of plane is ⊥ to both rays p & q.

∴ $l = \bar{p} \times \bar{q}$ } Cross product.

* (we will get the same result we had before)

ex: $\bar{p} = (5,2,1)$
$\bar{q} = (3,2,1)$

$$l = \begin{bmatrix} i & j & k \\ 5 & 2 & 1 \\ 3 & 2 & 1 \end{bmatrix}$$

$$= i(0) - j(2) + k(4)$$

$$= -2\hat{j} + 4\hat{k}$$

$$l = \begin{bmatrix} 0 \\ -2 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix}$$

(Scaling is fine)

---

ex: Ideal point of line $[0\ 1\ -2]^T$ is $[1\ 0\ 0]^T$.
This is same as $[0\ 1\ k]^T$ for any k.
→ line

ex: Line joining $[3\ 4\ 0]^T$

Point of intersection of 2 lines:

- $l^T x = m^T x = 0$, $x = l \times m$

(Duality → similar to line through 2 points)

---

*Note: Ideal point of $l = [a\ b\ c]^T$ is $[b\ -a\ 0]^T$

---

*Conics:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

- $ax^2 + bxy + cy^2 + dxw + eyw + fw^2 = 0$ (Using homogeneous coordinates)

↓

If $w = 1$, we get the original equation of the conic.

$$\begin{bmatrix} x & y & w \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0$$

$$x^T C x = 0$$

↑ Symmetric matrix

(Degenerate cases where C reduces conic to a line)

*Tangents:

$\Rightarrow$ $x^T (Cx) = 0$

$1\times3 \quad 3\times3 \; 3\times1$

$x^T l = 0$

$\boxed{l = Cx}$ Gives us a line through x. (So it touches the conic)

↓

Tangent to conic at x.

Proof: (Suppose l intersects C at another point y,

$y^T C y = 0$, $y^T l = 0 \Rightarrow y^T C x = 0$

$y^T C y = 0$

$\therefore l$ = Tangent

So the line l only intersects the conic at a single point. { Only possible if x & y are the same point. { $\Rightarrow$ If x, y lie on a line $ax + b$ (1-x) also lies on the line $\Rightarrow$ So it would satisfy C, so they would all lie on C

Dual Conic:

- [redacted] Set of all lines tangential to the conic C.

$l^T C^* l = 0$ & $l^T C^{-1} l = 0$ } Set of lines tangential to C

$\Rightarrow C^* =$ Dual of $C = C^{-1}$.

- Translation $\overset{(2)}{\subset}$ Euclidean $\overset{(3)}{\subset}$ Similarity $\overset{(4)}{\subset}$ Affine (6) $\subset$ Projective (8)

$\Uparrow$ (2) → DoF

$\Rightarrow$ Transformations.

---

* Projective Transforms: (Collinearity is preserved)

- $x' = Hx$ } Transformed points
- $l' = H^{-T} l$ } Transformed line
- $C' = H^{-T} C H^{-1}$, $C^{*'} = H C^* H^T$ } Transformed conics & duals.

Note: Isometric transforms are,

* $$\begin{bmatrix} s\cos\theta & -s\sin\theta & a \\ s\sin\theta & s\cos\theta & b \\ 0 & 0 & 1 \end{bmatrix} \quad s = \pm 1$$

(If s = Any non zero scalar → we have scaling as well).

(Similarity transform)

↓

Preserve angles, parallelism, ratio of lengths & circular points.

Note: Metric structure is the geometric reconstruction of an object that is defined upto an unknown similarity transformation. (Reconstructions are metric structures)

(True scale, rotation etc are not known just from image)

Note: Affine transforms map points at ∞ to other points at ∞.

$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow$ Affine transforms.

Projective Transforms decomposition: (No need to know how)

- $\begin{bmatrix} A & t \\ v^T & 1 \end{bmatrix} = H_P \cdot H_A \cdot H_S = \begin{bmatrix} I & 0 \\ v^T & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} SR & \\ 0 & \end{bmatrix}$

$K =$ Upper Δ las with $\det = 1$

* • Preserves cross ratios & collinearity.
- 8 degrees of freedom (4 point matches needed)
- Finite points can map to ideal points & vice-versa (train tracks)

*

* **Cross Ratios on a line:**

$$\text{Cross}(x_1, x_2, x_3, x_4) = \frac{|x_1 x_2||x_3 x_4|}{|x_1 x_3||x_2 x_4|}$$ where $|xy|$ = 1D signed distance ↑ along line $= \det\begin{pmatrix} x_1 & \\ x_2 & \end{pmatrix}$

(For 4 concurrent, coplanar lines, cross ratio of 4 points measured as intersections of any line with those lines is constant)

---

* **Circular points:** (Preserved till similarity transforms)

* • Intersection of circle with line at infinity.

$\Rightarrow ax^2 + ay^2 + dxw + eyw + fw^2 = 0$

$\Rightarrow$ Intersection with line at $\infty \Rightarrow w = 0$

$\Rightarrow ax^2 + ay^2 = 0$

$\Rightarrow x^2 + y^2 = 0$

$\Rightarrow$ Simple solutions are $(1, i) \Rightarrow \begin{bmatrix} 1 \\ i \\ 0 \end{bmatrix}$ and $(1, -i) \Rightarrow \begin{bmatrix} 1 \\ -i \end{bmatrix}$

Note: Table comparing properties of all transforms in slides.

Finding Line at Infinity: (From images)

Graph Based Methods: (in CV)

- Irregularly sampled data domains are generally modelled as graphs.
- BSP (Bulk Simulated Processing) → Local graph updates!

⇒ Spectral Graph Theory: Function defined on vertices. Smooth this function etc using the structure of the graph.

- MRF based methods also build a graph.
- Graphs are non euclidean so we can't compute derivatives etc.

⇒ We want our edges to be as local as possible over the euclidean manifold.

* • Graph Laplacian: $L = D - U$ (Embedding)

* ↳ $-\exp(\|v_i - v_j\|^2/2) = w_{ij}$ } Set weights of edges as such.

$L = U \Lambda U^T$ } Ignore zero eigen values & its corresponding eigen vectors

- The eigen vectors are then used to project graph nodes onto the euclidean space spanned by the d eigen vectors.
(Embedding space) (Graph Laplacian Embedding)

→ The distance in this embedding space depicts the average connectivity in the original graph.

---

* Heat (Diffusion) Kernel Embedding:

$H(t) = e^{-tL} = U e^{-t\Lambda} U^T$ (L = Graph Laplacian)

$H(t)$: Kernel Matrix. ($n \times n$ → n nodes in graph)

$e^{-t\Lambda}$: Diagonal matrix

$U e^{-\frac{t}{2}\Lambda} \cdot \left(e^{-\frac{t}{2}\Lambda}\right)^T \cdot U^T$

$= X \cdot X^T$

- The i^{th} column of the matrix at a time t shows how the heat source at i spreads if rest are treated as the sinks.

(Similar to how Adjacency matrix shows one edge connectivity, $A^2 \Rightarrow$ Two edge connectivity b/w nodes and so on...)

---

*Dense 3D Matching: (Paper) (Using Heat Diffusion)

* $\Rightarrow$ Topology (Structure) changes in the graph give us different graph embeddings so comparisions are hard.
  - ↳ Heat diffusion at small scales however works, so we can use this.

- Given certain reference points on a graph, we can use them to compute a heat diffusion based descriptor for every other point on the graph.
  (Similar to barycentric point descriptors in 2D)

$\Rightarrow$ So using these point matches, we find a dense matching between all nodes using the descriptors. We then apply EM for smoothing.

---

* Spectral Correspondence for Image Matching: (Paper) (CVPR)

- Take 2 images → compute SIFT per pixel.
  - ↳ Graph b/w pixels that are similar
  - ↳ Also connect pixels from image 1 to image 2 based on similarity.

$\Rightarrow L = D - A = U \Lambda U^T$

- Each eigen vector is of shape $(n_1 + n_2) \times 1$ (No. of pixels in Image 1 + No. of pixels in Image 2)
  - ↳ Split into $(n_1 \times 1, n_2 \times 1)$
  - ↳ Compute similarity between these, matching can also be done using SIFT here on these eigen vectors.

Uses MDER ←

* Image Annotation: (Paper)

*

- Uses Hypergraph Laplacian
  - Edges can connect multiple nodes.
  - (We can represent a hyperedge using multiple dyadic edges)

$$\Pi = [he_1, \ldots, he_p]$$

- Some Laplacian is defined here
- Each image in dataset is a node here.
  - ↳ Feature space (VGG features) used to create a k-neighbour graph.
- Hypergraph heat diffusion to diffuse labels from training image nodes to test image nodes.
  (Initially test nodes are connected to all with zeros as label values)

⇒ Useful since for more occuring labels we use small diffusion size and larger diffusion size for less frequent labels

- ↳ This was an issue not addressed by earlier methods.
