CS4186 notes

1. Image filtering

Image

- A grayscale image: a function f from $\mathbb{R}^2$ to $\mathbb{R}$, where $f(x, y)$ gives the intensity at position (x, y).
- A digital image is a discrete (sampled and quantized) version of this function, typically stored as a matrix of intensity values (0 = black, 255 = white).
- Operators can be applied to images (transformation):

$$g(x, y) = f(x, y) + 20$$

Filters

- **Purpose of Filtering**:
  - Extract useful information (e.g., edges, contours).
  - Enhance images (e.g., remove noise, sharpen).
- **Linear Filtering (Convolution)**:
  - Replace each pixel with a weighted sum of its neighbors.
  - The weights are defined by a **kernel** (or mask/filter).

Cross-Correlation and Convolution

- **Cross-Correlation ($G = H \otimes F$)**:

$$G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v]$$

- **Convolution ($G = H * F$)**:

$$G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v]$$

  The kernel is flipped horizontally and vertically compared to cross-correlation.

Mean Filtering

- A simple linear filter that replaces each pixel with the average of its neighbors (moving average).
- Used for noise reduction but may blur high-frequency signals.
- example filter H:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Gaussian Filter

- **Gaussian Kernel**:

$$G_\sigma = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

  The kernel weights are normalized to sum to 1.

- **Properties**:
  - Removes high-frequency components (low-pass filter).
  - Convolution of a Gaussian with itself yields another Gaussian.
  - Less disruptive to high-frequency signals compared to mean filtering.

Sharpening

- Achieved by adding back the high-frequency details removed by blurring:

$$\text{Sharpened} = \text{Original} + \alpha \times \text{Detail}$$

  where **Detail** = Original - Blurred.

Thresholding

- A non-linear operation to convert grayscale images to binary:

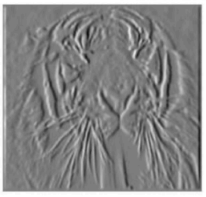
$$g(m,n) = \begin{cases}255 & \text{if } f(m,n) > A\\ 0 & \text{otherwise}\end{cases}$$

2. Edge Detection

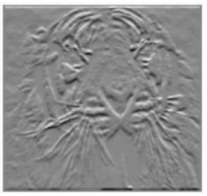
- **Goal**: Identify visual changes (discontinuities) in an image, which encode semantic information.
- **Purpose**: Convert a 2D image into a set of curves for compact feature representation.

Image Derivatives

Image f



$\frac{\partial f(x,y)}{\partial x}$



$\frac{\partial f(x,y)}{\partial y}$

- **Finite Difference Approximation**:
  - Partial derivative in x-direction:

$$\frac{\partial f}{\partial x}[x,y] \approx F[x+1,y] - F[x,y]$$

  - Partial derivative in y-direction:

$$\frac{\partial f}{\partial y}[x,y] \approx F[x,y+1] - F[x,y]$$

- Implemented using linear filters (kernels H_x and H_y).

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

$$H_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Image Gradient

- **Gradient Vector**:

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

  - Points in the direction of most rapid intensity increase.
- **Gradient Magnitude** (edge strength):

$$\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$$

- **Gradient Direction**:

$$\theta = tan^{-1}(\frac{\partial f}{\partial y}/\frac{\partial f}{\partial x})$$

Smoothing and Differentiation

- **Noise Sensitivity**: Derivatives amplify noise. Solution: Smooth the image first.
- **Associative Property of Convolution**

$$\frac{d}{dx}(f * h) = f * \frac{d}{dx}h$$

  - Smoothing (h) and differentiation can be combined into a single kernel.

Sobel Operator

- Approximates the derivative of Gaussian.
- **Kernels**:
  - $x$$x$direction ($S_x$):

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

  - $y$$y$direction ($S_y$):

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

- Normalization (optional $\frac{1}{8}$ scaling) ensures correct gradient magnitude.

Criteria for Good Edge Detection

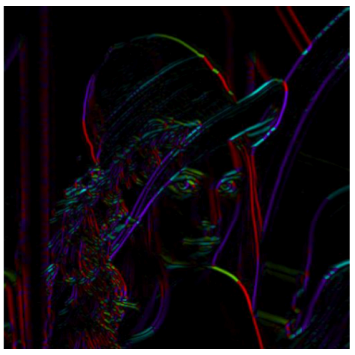
1. **Good Detection**: Minimize false negatives (miss real edges) and false positives (ignore noise).
2. **Good Localization**: Detected edges should be close to true edges with minimal duplication.

Canny Edge Detector

1. **Filter** image with derivative of Gaussian:

2. Compute **gradient magnitude** and **orientation**.
    - **gradient magnitude:**

    - **orientation:**



3. Apply **non-maximum suppression** to thin edges.

   - Preserve local maxima in gradient direction, suppress others.
4. Perform **thresholding** (hysteresis) to link edges.

   - Use high/low thresholds to distinguish strong/weak edges.

3. Image Resampling

Image Subsampling

- **Definition**: Reducing image size by discarding pixels (e.g., throwing away every other row and column).
- **Problem**: loss of high-frequency details, artifacts like moiré patterns.

Gaussian Pre-Filtering

- **Purpose**: Prevent aliasing during subsampling by smoothing the image before reducing its resolution, also help to build Gaussian Pyramid.

- **Method**:
    1. Apply a **Gaussian filter** to blur the image and remove high-frequency components.
    2. **Subsample** the smoothed image (e.g., take every 2nd pixel for half-size).

4. Color and Texture

Image Histograms (Grayscale)

- **Definition**:
    - A histogram $H[i]$ counts the number of pixels with intensity i in a grayscale image.
    - **Normalized histogram**: $P[i]$ represents the percentage of pixels with intensity i.
- **Mathematical Formulation**:

$$h(i) = \sum_x \sum_y \begin{cases} 1 & \text{if } F(x,y) = i \\ 0 & \text{otherwise} \end{cases}$$

Color Histograms

- **Purpose**: Represent the distribution of pixel values in color images for tasks like database query or classification.
- **Methods**:
    1. **3D Histogram**: Combine R, G, B channels into a single 3D histogram.
    2. **Concatenated Histograms**: Create separate histograms for each channel (R, G, B) and concatenate them.
    3. **Pseudo-Color Encoding**:
        - Encode color into a single value (e.g., 3 bits R, 3 bits G, 2 bits B → 8-bit pseudo-color).
    4. **Normalized Color Space**: Use histograms in perceptually uniform spaces (e.g., HSV, Lab).

5. Image Texture

Edge-Based Texture Measures

1. **Edge Density**:
    - Measures "busyness" of a region by counting edge pixels.
    - Formula:

$$F_{\text{edgeness}} = \frac{|\{p \mid \text{gradient magnitude}(p) > \text{threshold}\}|}{N}$$

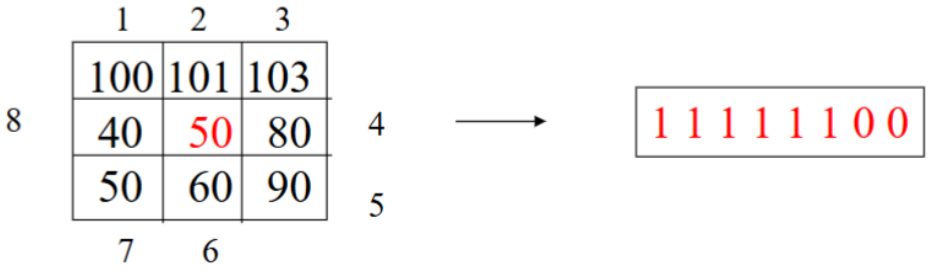
where N is the area size.

2. **Edge Magnitude/Direction Histograms**:
   - Represents texture via histograms of gradient magnitudes ($H_{\text{magnitude}}$) and directions ($H_{\text{direction}}$).
   - Feature vector:

$$F_{\text{magdir}} = (H_{\text{magnitude}}, H_{\text{direction}})$$

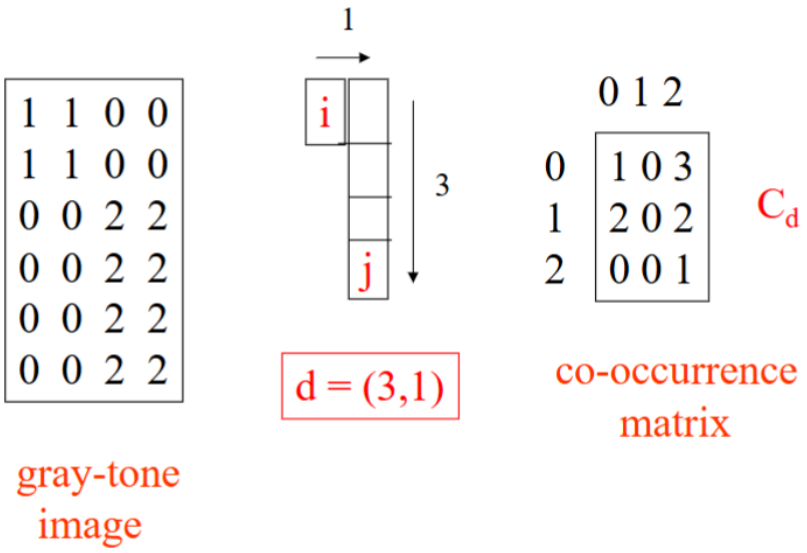
Local Binary Pattern (LBP)

- **Principle**: Encodes local texture by comparing a pixel's value with its neighbors.



Co-Occurrence Matrix Features

- **Definition**: A matrix $C_d(i, j)$ counts how often pixel values i and j co-occur with spatial offset $d = (dr, dc)$.



- **Challenge**: Selecting optimal spatial offset d*d* (scale-dependent).

6. Feature Extraction

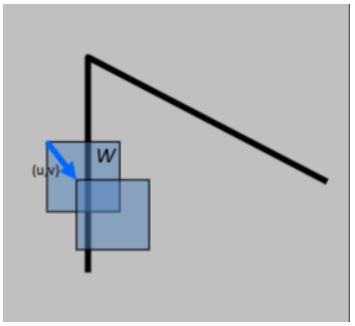
- **Purpose**: Identify distinctive points (corners, blobs) for tasks like panorama stitching, 3D reconstruction, and object recognition.
- **Key Properties of Good Features**: Locality, Distinctiveness and Invariance

Corner Detection Principle

- **Definition**: A corner is the intersection of two edges, where the image gradient changes significantly in all directions.
- **Local Uniqueness Test**:
  - **Flat region**: Minimal change in any direction.
  - **Edge**: Change along one direction (gradient perpendicular to edge).
  - **Corner**: Large change in all directions.

Harris Corner Detection

1. **SSD Error for Window Shift**:



   - We are happy if this error is high
   - For a window W shifted by (u, v):

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

   - Small motion approximation (Taylor expansion):

$$I(x+u, y+v) \approx I(x,y) + I_x u + I_y v$$

   where $I_x = \frac{\partial x}{\partial I}, I_y = \frac{\partial I}{\partial y}$.

2. **Error Surface Approximation**:

$$(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

where:

$$A = \sum_W I_x^2, \quad B = \sum_W I_x I_y, \quad C = \sum_W I_y^2.$$

- Matrix $H = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$ captures local image structure.

3. **Eigenvalue Analysis**:
   - Solve $\det(H - \lambda I) = 0$ for eigenvalues λ_1, λ_2.
   - **Corner Response**:

     $$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \quad (\text{Harris operator})$$

     Alternatively:

     $$f = \frac{\det(H)}{\text{trace}(H)} = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

   - **Interpretation**:
     - **Corner**: λ_1, λ_2 large, $\lambda_1 \sim \lambda_2$, R large.
     - **Edge**: One eigenvalue $\gg$ the other, $R < 0$.
     - **Flat**: λ_1, λ_2 small.

Harris Algorithm Steps

1. Compute gradients I_x, I_y (e.g., using Sobel filters).
2. Construct matrix H for each pixel:

   $$H = \begin{bmatrix} \sum_W I_x^2 & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y^2 \end{bmatrix}$$

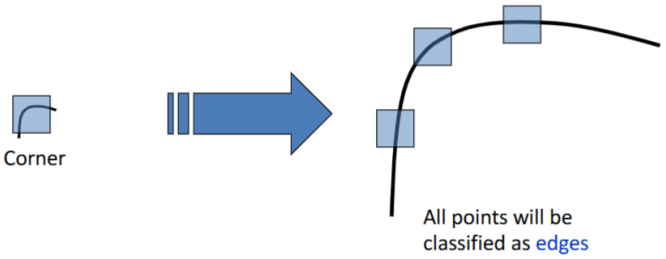
3. Compute corner response R or f.
4. Threshold R and perform non-maximum suppression to isolate corners.

Harris Detector Invariance Properties

- **Rotation Invariance**
  - Eigenvalues unaffected by image rotation.
- **Translation Invariance**:
  - Derivatives and window functions are shift-invariant.
- **Affine Intensity Invariance**:

- Invariant to intensity shift $I \rightarrow I + b$
- partially invariant to scaling $I \rightarrow aI$ (uses derivatives).

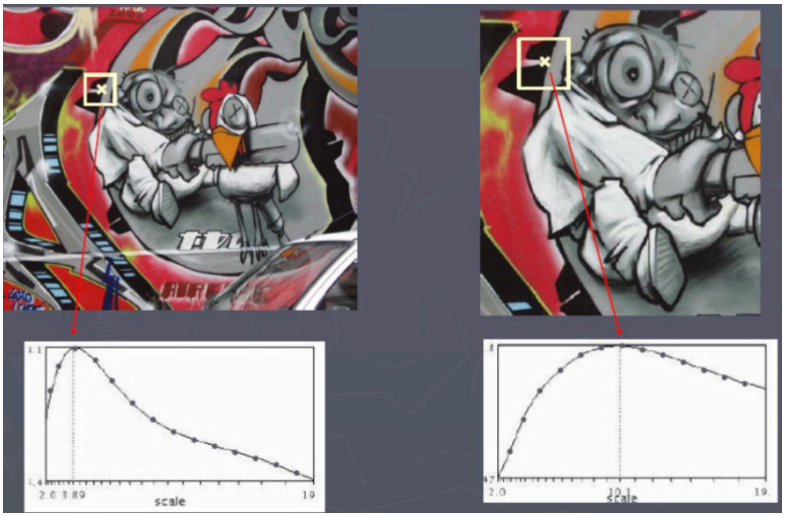
- **Scale Non-Invariance**:



  - Fails under significant scaling (corners may be misclassified as edges).

Scale-Invariant Detection

- **Key Idea**: Find local maxima of a response function f (e.g., Harris operator) in **both position and scale**.
- **Automatic Scale Selection**:
  - Search for scale σ that maximizes $f(x, \sigma)$.



- **Implementation**:
  - Use a **Gaussian pyramid** to handle multiple scales efficiently.

Laplacian of Gaussian (LoG)

- **Blob Detection**:
  - LoG identifies blobs as local maxima/minima in scale-space.
- **Operator Definition**:

$$\nabla^2 h_\sigma(u,v) = \frac{\partial^2 h_\sigma}{\partial x^2} + \frac{\partial^2 h_\sigma}{\partial y^2},$$

where $h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$ (Gaussian kernel).

- **Scale-Space Extrema**:
  - Detect blobs by finding peaks in $\nabla^2 h_\sigma \star I$ across scales σ.

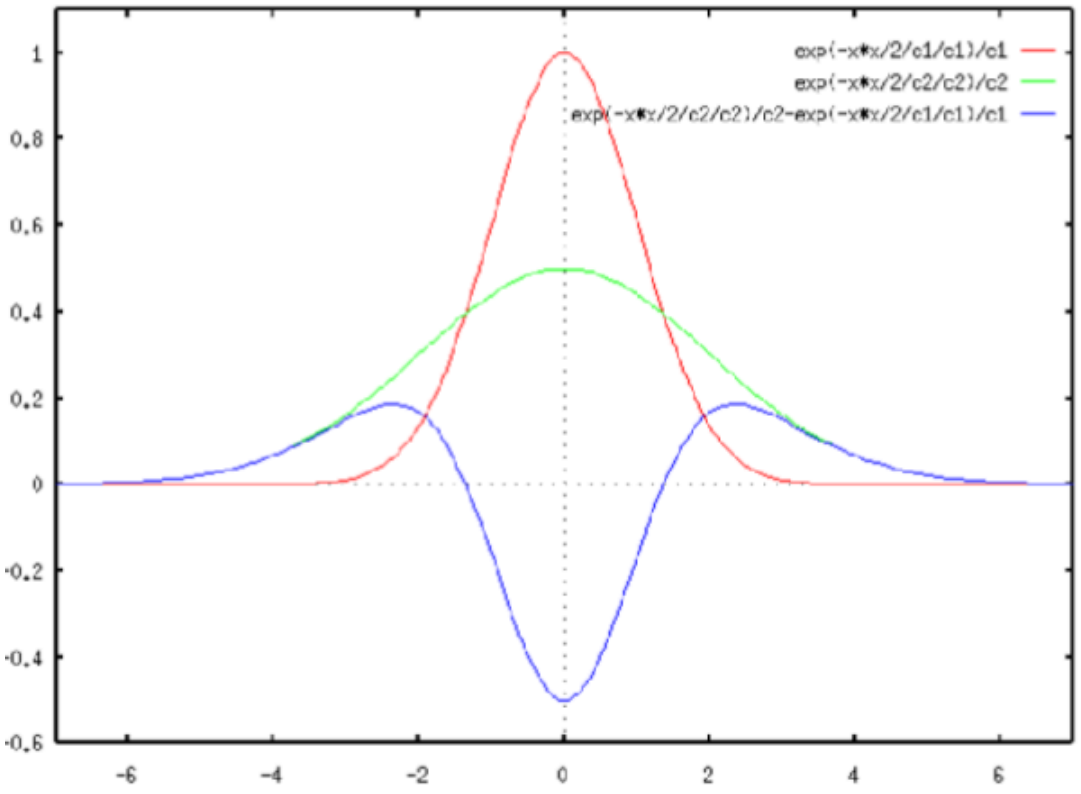
Feature Descriptors

- **Purpose**: Match detected points between images.
- **Approaches**:
  - **Window Matching**: Compare pixel neighborhoods.
  - **SIFT (Scale-Invariant Feature Transform)**:
    - Robust to rotation, scale, and illumination changes.

LoG-DoG (Difference of Gaussian)

- **Approximation**:

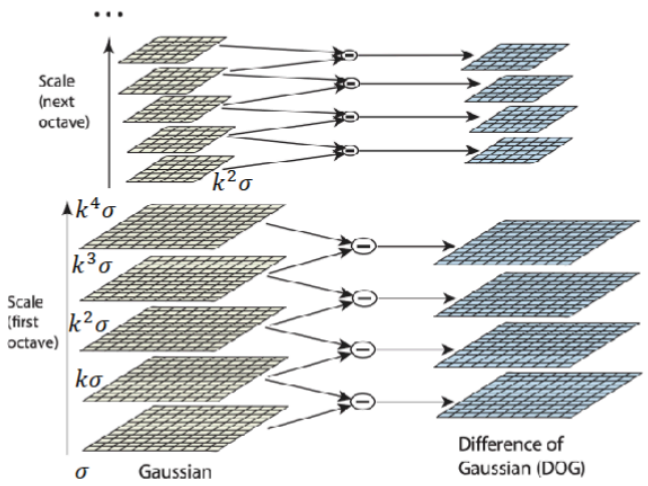
  The Laplacian of Gaussian (LoG) can be approximated by the **Difference of Gaussians (DoG)** at two scales σ_1 and σ_2 (blue below):



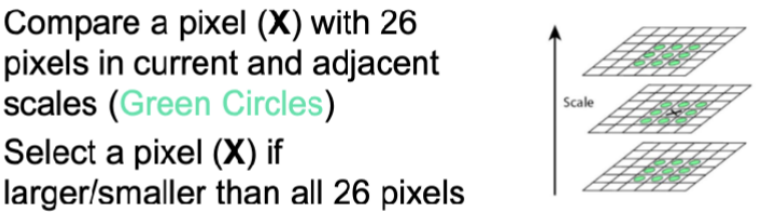
Scale Invariant Feature Transform (SIFT)

Key Steps:

1. **Scale-Space Peak Selection**:
    - Construct a Gaussian pyramid (multiple octaves and scales).



    - Detect local maxima/minima in **DoG** space across scales and positions.



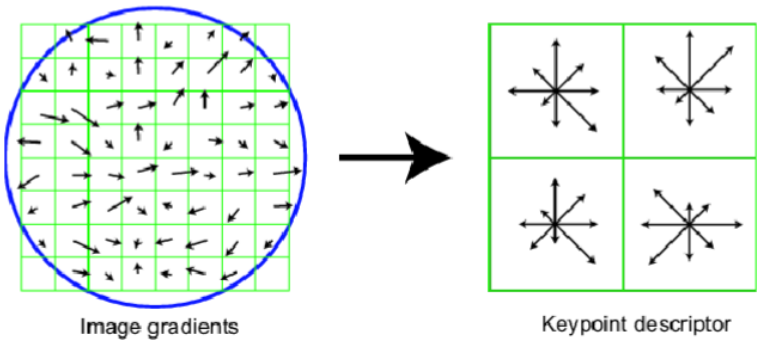
2. **Keypoint Localization**:
    - Refine candidate keypoints by eliminating low-contrast or edge responses.
3. **Orientation Assignment**:
    - Compute gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ at the keypoint scale:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

    - Assign dominant orientation(s) using a **36-bin histogram** weighted by gradient magnitude.
4. **Descriptor Generation**:
    - Divide a **16×16** window around the keypoint into **4×4 cells**.

- Compute an **8-bin orientation histogram** per cell (relative to keypoint orientation).
- Concatenate histograms into a **128-dimensional vector** (normalized for illumination invariance).



Feature Matching

- **Distance Metrics**:
  - **SSD (Sum of Squared Differences)**:

$$d(f_1, f_2) = \|f_1 - f_2\|^2$$

  - **Ratio Test**:

$$\text{ratio} = \frac{\|f - f_1\|}{\|f - f_2\|}$$

    where f_1 is the best match and f_2 is the second-best match in the target image.
    - Rejects ambiguous matches (ratio > threshold).

7. Image Transformations

Image Warping vs. Filtering

- **Image Filtering**: Modifies pixel values (range transformation):

$$g(x) = h(f(x))$$

- **Image Warping**: Modifies pixel coordinates (domain transformation):

$$g(x) = f(h(x))$$

Forward vs. Inverse Warping

- **Forward Warping**:

- Maps source pixels $f(x, y)$ to target coordinates $(x', y') = T(x, y)$.

- **Inverse Warping**:
  - Samples target pixels $g(x', y')$ from source coordinates $(x, y) = T^{-1}(x', y')$

Parametric (Global) Transformations

- Represented as $p' = T(p)$, where T is a matrix.
- **Linear Transformations (2×2 Matrix)**:
  - Rotation:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

  - Scaling:

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

- **Limitation**: 2×2 matrices **cannot** represent translation.

Homogeneous Coordinates

- **Purpose**: Unify linear transformations and translation in 3D space.
- **Representation**:

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \text{Conversion: } \begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

- **Translation**:

$$\mathrm{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Affine Transformations

- **Definition**: Combines linear transformations and translation.

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

8. Image Alignment

Image Alignment Problem

- **Goal**: Compute a transformation T that aligns two images (A and B) given a set of feature matches.
- **Approach**:
  1. Detect and match features between images.
  2. Estimate T using matched points (e.g., translation, homography).

Simple Case: Translation Estimation

- **Displacement Calculation**:

  For n matches, compute average translation (x_t, y_t):

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

- **Least Squares Formulation**:

  Minimize sum of squared residuals:

$$C(x_t, y_t) = \sum_{i=1}^{n} [(x_i + x_t - x_i')^2 + (y_i + y_t - y_i')^2]$$

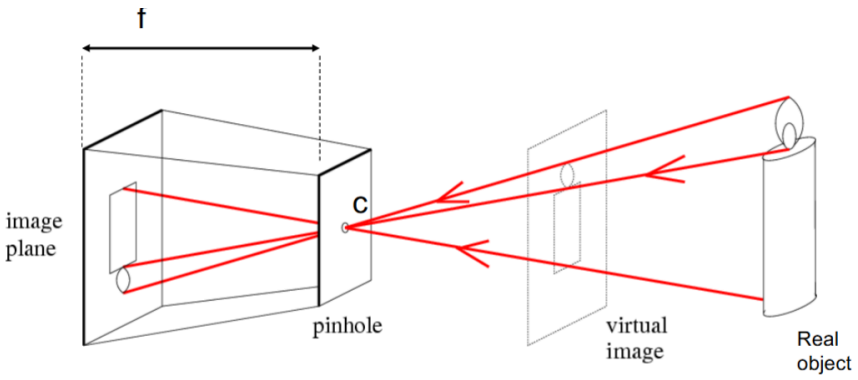
General RANSAC Algorithm (Handling Outliers)

1. **Sample**: Randomly select s points (minimal set to fit the model).
2. **Fit**: Compute model parameters (e.g., line, homography).
3. **Score**: Count inliers (points within error tolerance).
4. **Repeat**: N times, keeping the best model.
5. **Optimize**: Final least squares fit on all inliers.

9. Camera

Pinhole Camera Model

- **Principle**: Light rays pass through a small aperture (pinhole) to form an inverted image on the image plane.



f = Focal length
c = Optical center of the camera

- **Projection Equations**:

  For a 3D point $P = (x, y, z)$, its 2D projection $P' = (x', y')$ is:

  $$x' = f\frac{x}{z}, \quad y' = f\frac{y}{z}$$

  where f is the focal length (distance from pinhole to image plane).

Homogeneous Coordinates

- **Purpose**: Simplify transformations using matrix operations.
- **Conversion**:
  - 3D point:

    $$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  - 2D point:

    $$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Normalization**:

  $$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (\frac{x}{w}, \frac{y}{w})$$

Camera Coordinate Systems

1. **World Coordinates** P_w: 3D scene points (x_w, y_w, z_w).

2. **Camera Coordinates** P_c: 3D points (x, y, z) relative to the camera's optical center.
3. **Image Coordinates** P': 2D points (x', y') on the image plane.

Intrinsic Parameters

- **Camera Matrix** K: Maps camera coordinates to pixel coordinates.

$$K = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

  where:

  - $\alpha = fk$ (horizontal scale factor, k: pixels per unit length).
  - $\beta = fl$ (vertical scale factor, l: pixels per unit length).
  - (c_x, c_y): Principal point (image center).

- **Projection with Intrinsics**:

$$P' = MP_c = K \begin{bmatrix} I & 0 \end{bmatrix} P_c = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \left(\alpha \frac{x}{z} + c_x, \beta \frac{y}{z} + c_y\right)$$

Extrinsic Parameters

- **Transformation**: Converts world coordinates to camera coordinates.

$$P_c = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} P_w$$

  where:

  - R: 3×3 rotation matrix.
  - T: 3×1 translation vector.

Projection Matrix (Combined Intrinsic + Extrinsic)

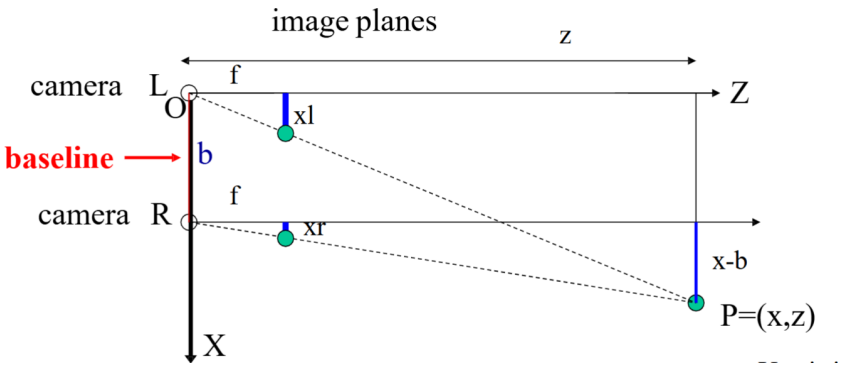
- **Full Projection**:

$$P' = K \begin{bmatrix} R & T \end{bmatrix} P_w$$

10. Stereo Vision

Depth from Stereo

- **Principle**: Disparity (horizontal shift) between corresponding points in two images is inversely proportional to depth.



- **Depth Formula**:

$$z = f \cdot \frac{b}{d}$$

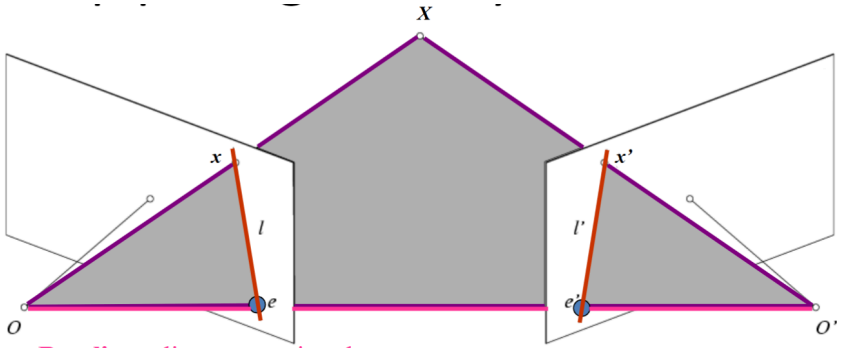
where:

  - z: Depth of the 3D point.
  - f: Focal length.
  - b: Baseline (distance between cameras).
  - $d = x_l - x_r$: Disparity (pixel difference).

- **3D Point Reconstruction:**

$$x = \frac{x_l \cdot z}{f}, \quad y = \frac{y_l \cdot z}{f}$$

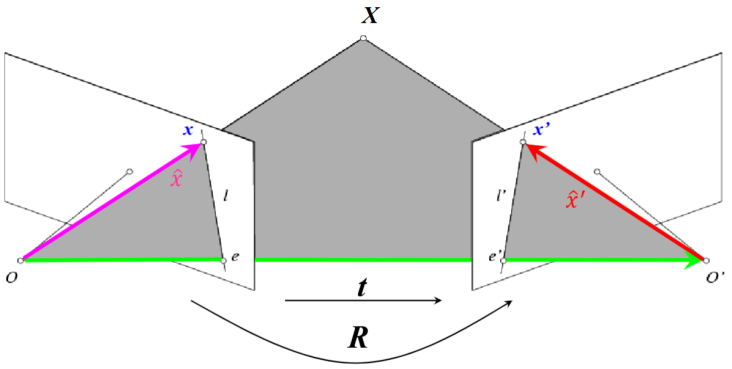
Epipolar Geometry



  - **Key Terms**:
    - **Baseline**: Line connecting camera centers (pink line).
    - **Epipoles**: Intersections of baseline with image planes (e & e').

- **Epipolar Plane**: Plane containing baseline and a 3D point (grey area).
- **Epipolar Lines**: Intersections of epipolar plane with image planes (l & l').

- **Epipolar Constraint**: Corresponding points must lie on conjugate epipolar lines.
- **Calibrated Case:**

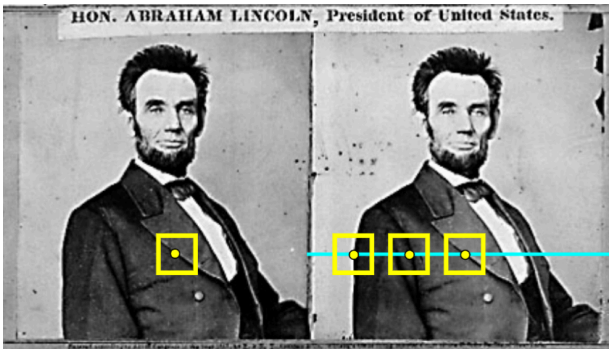


  - **Assumptions**:
    - Camera intrinsics (K) and extrinsics (R, t) are known.
    - World coordinates align with the first camera's frame.
  - **Projection Matrices**:
    - Camera 1: $P_1 = [I \mid 0]$.
    - Camera 2: $P_2 = [R \mid t]$.
  - **Normalized Coordinates**: Remove camera intrinsics via $\hat{\mathbf{x}} = K^{-1}\mathbf{x}$.
  - **Essential Matrix (E)**
    - **Derivation**: For corresponding points $\hat{x}$ (Camera 1) and $\hat{x}'$ (Camera 2), the vectors $R\hat{x}$, t, and $\hat{x}'$ are coplanar.
    - **Epipolar Constraint**:

$$\hat{x}'^{\top} E \hat{x} = 0, \quad \text{where } E = [t_\times]R$$

    - **Properties**:
      - $E\hat{x}$ defines the epipolar line l' in the second image.
      - E encodes the relative pose (R, t) between cameras.

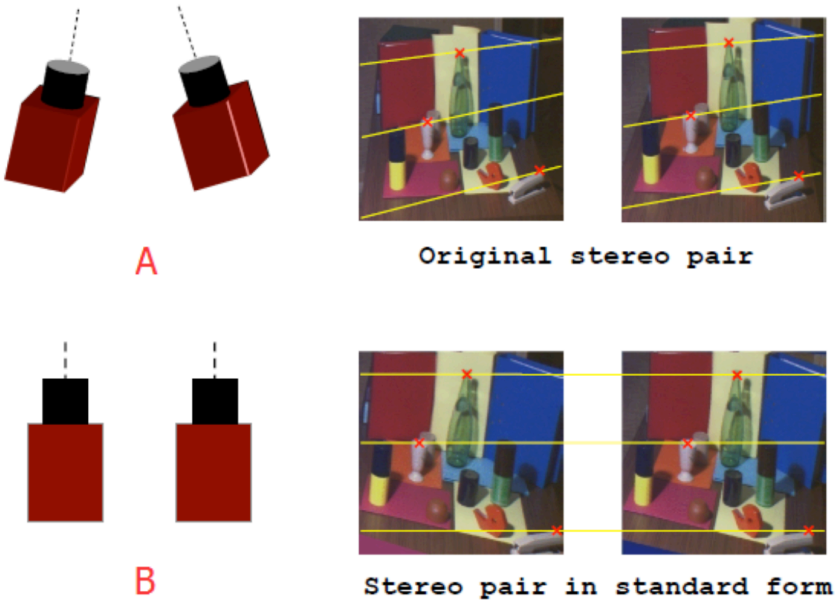
Correspondence Problem



- **Challenge**: Find matching points x (left image) and x' (right image).
- **Constraints**:
  - **Uniqueness**: Each point has at most one match.
  - **Ordering**: Matches preserve left-right order (except occlusions).
  - **Smoothness**: Disparity varies smoothly except at depth boundaries.

Stereo Rectification

- **Goal**: Align epipolar lines with image scanlines for simplified search.
- **Process**: Transform images so that corresponding points lie on the same horizontal line.



Matching Algorithms

- **Window-Based Matching**:
  - Compare patches using similarity measures:
    - **Sum of Absolute Differences (SAD)**:

$$\sum_{(i,j)\in W} |I_1(i,j) - I_2(x+i, y+j)|$$

- **Sum of Squared Differences (SSD)**:

$$\sum_{(i,j)\in W} (I_1(i,j) - I_2(x+i, y+j))^2$$

- **Normalized Cross-Correlation (NCC)**:

$$\frac{\sum_{(i,j)\in W} I_1(i,j) \cdot I_2(x+i, y+j)}{\sqrt{\sum_{(i,j)\in W} I_1^2(i,j) \cdot \sum_{(i,j)\in W} I_2^2(x+i, y+j)}}$$

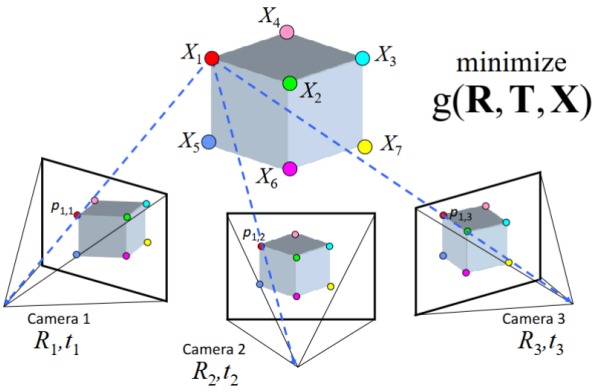
- **Trade-offs**:
  - Small windows: More detail but noisy.
  - Large windows: Smoother but lose edges.

Stereo Pipeline

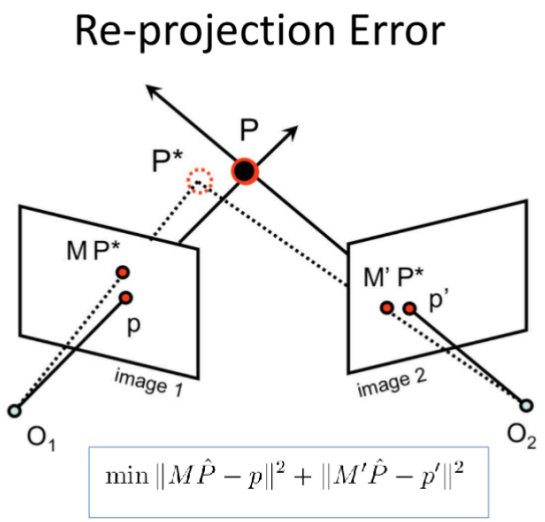
1. **Calibrate Cameras**: Estimate intrinsic/extrinsic parameters.
2. **Rectify Images**: Align epipolar lines.
3. **Compute Disparity**: Search for matches along scanlines.
4. **Estimate Depth**: Convert disparity to depth using $z = f \cdot \frac{b}{d}$.

11. Structure from Motion (SfM)

- **Goal**: Reconstruct 3D scene structure and camera poses from multiple images.
- **Input**: Images with corresponding 2D feature points.
- **Objective**: Minimize reprojection error.



Reprojection Error

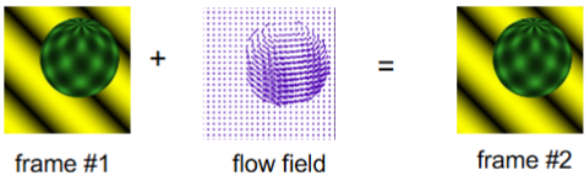


Key Challenges

1. **Correspondence Estimation**:
   - Detect features (e.g., SIFT) and match across images.
   - Use RANSAC to estimate fundamental matrices between pairs.
2. **Scale of Problem**:
   - Large-scale SfM involves thousands of images and millions of points.
   - Sparse matrices and iterative solvers (e.g., Levenberg-Marquardt) are used.

12. Optical Flow and Motion Estimation

Optical Flow Definition



- **Optical Flow**: A vector field representing the apparent motion of objects between consecutive frames, caused by relative motion between the observer and the scene.
- **Goal**: Estimate the displacement (u, v) for each pixel from $I(x, y, t)$ to $I(x, y, t + 1)$.

Key assumptions of Lucas-Kanade Tracker

- **Brightness constancy**: projection of the same point looks the same in every frame
- **Small motion**: points do not move very far

- **Spatial coherence**: points move like their neighbors

Brightness Constancy Constraint

- **Assumption**: Pixel intensity remains constant between frames:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

- **Linearized Form** (via Taylor expansion):

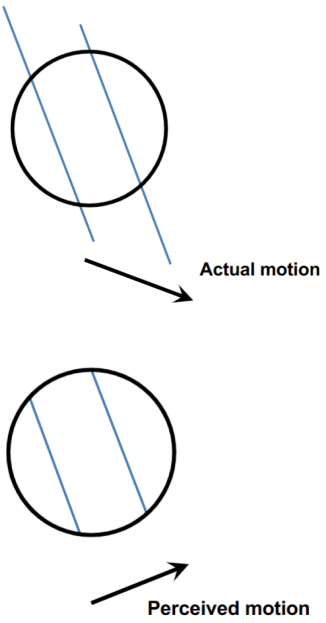
$$I_x u + I_y v + I_t = 0 \quad or \quad \nabla I \cdot \begin{bmatrix} u & v \end{bmatrix}^T + I_t = 0$$

  where:

  - I_x, I_y: Spatial derivatives.
  - I_t: Temporal derivative.

Aperture Problem

- **Ambiguity**: Motion along edges is underconstrained (only normal flow can be estimated).



- **Solution**: Use spatial coherence (Lucas-Kanade method) or multi-scale approaches.

Lucas-Kanade Method

Assume the pixel's neighbors have the same (u,v)

– If we use a 5x5 window, that gives us 25 equations per pixel

$$0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u\ v]$$

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

- Overconstrained linear system

$$\underset{25\text{x}2}{\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix}} \underset{2\text{x}1}{\begin{bmatrix} u \\ v \end{bmatrix}} = - \underset{25\text{x}1}{\begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix}} \qquad \underset{25\text{x}2}{A}\ \underset{2\text{x}1}{d} = \underset{25\text{x}1}{b}$$

Least squares solution for *d* given by $(A^T A)\ d = A^T b$

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}$$

Challenges and Solutions

1. **Large Motion**:
   - **Coarse-to-Fine Estimation**: Use Gaussian pyramids to handle large displacements.
     - Estimate flow at low resolution, then refine at higher resolutions.
2. **Iterative Refinement**:
   - Warp $I(t)$ toward $I(t+1)$ using current flow estimate.
   - Recompute residuals and update flow iteratively.