

05. Segmentation

- Goal: Separate image into coherent regions
- Idea: **Clustering** - Group similar data points together
- Challenges: What makes 2 points same/different? Choice of features (e.g. Color, Intensity, Position), Which clustering algorithm?
- k-Means Clustering** - Iteratively re-assign points to nearest cluster center
 - Randomly initialize the cluster centers c_1, \dots, c_K
 - For each point p_i , find the closest c_j to put p_i in
 - Set c_j to be mean of points in cluster j
 - Repeat, if c_j have changed up to some threshold
 - Pros: Simple, Converges to local min.
 - Cons: Setting K , Sensitive to initial centers (Since k-means converges to local min.), Sensitive to outliers (Can add more clusters), Assumes spherical clusters
- Simple Linear Iterative Clustering (SLIC) Superpixels**
 - Superpixel** - Group of pixels that share common traits
 - Application: Inputs to other CV algo. since more compact representation with perceptual meaning
 - Num. of pixels: n_{tp} ; Target num. of superpixels: n_{sp}
 - Initial width of each superpixel: $s = \sqrt{n_{tp}/n_{sp}}$
 - Features: $z = [r, g, b, x, y]$
 - Color distance: $d_c = ||\langle r_j, g_j, b_j \rangle - \langle r_i, g_i, b_i \rangle||$
 - Spatial distance: $d_s = ||\langle x_j, y_j \rangle - \langle x_i, y_i \rangle||$
 - Scaling factors: d_{cm} and $d_{sm} = s$ set as max. expected values of d_c and d_s respectively
 - $D = \sqrt{(\frac{d_c}{d_{cm}})^2 + (\frac{d_s}{d_{sm}})^2} = \sqrt{d_c^2 + (\frac{d_s}{s})^2} c^2$
 - Split img. into grid of size $s \times s$. Set cluster centers as lowest gradient position in 3×3 neighborhood from superpixel center to speed up convergence since initialize on value common to surrounding.
 - For each cluster center, check distance to all pixels within $2s \times 2s$ neighborhood. Assign pixels to closest checked center.
 - Update cluster centers using mean and repeat if not converged (Same as k-Means)
 - Optional: Replace superpixel region by average value to create stained glass effect
 - Modification of k-Means: Not random initialization, Compute pixel's distance only to closest set of cluster centers
- Mean-Shift Clustering** - Find local density maxima in feature space
 - Attraction basin** - Region in feature space for which all trajectories of centroids lead to same mode
 - Cluster** - All data points in attraction basin of a mode
 - For each data point:
 - Define window around and get centroid
 - Shift window to centroid
 - Repeat until window centroid stops moving
 - Segmentation with Mean Shift: Do mean shift and merge pixels in same attraction basin
 - Choosing window size: Trial and error, Sample points and use avg. dist. to knn. (Num. of neighbors needs

to be large enough to ensure increase in density)

- Larger window size \rightarrow Fewer clusters
- Pros: No assumptions on cluster shape, 1 parameter, Finds variable num. of modes (vs. specified k in k-Means), Robust to outliers
- Cons: Choosing h , Slow, Scales poorly with feature space dimension
- Optimizations:
 - After each run of mean shift, assign all points within radius r of end point to same cluster
 - Assign points in radius $c < r$ of search path to mode

06. Texture

- Texture** - Pattern with repeating elements
- Filter Bank** - Measures variety of structures in local neighborhood and generates multi-dimensional features
 - Goal: How to represent texture?
 - Idea: Apply filters with small windows to generate statistics that summarize local patterns. Dist. in feature space bet. windows \rightarrow Pixel's texture similarity
 - d filters $\rightarrow d$ -dimensional feature vector
 - Choosing window size: Try many sizes and look for one where statistic does not change much
 - Choose filters in different scales and orientations (to solve window size problem)
- Gabor Filter** - Represent filter banks mathematically by combining sinusoids with exp. (Gaussian) envelope
- Texton** - Characterizes texture by replacing each pixel with integer representing **texture type**
 - Apply filter bank to training image
 - Cluster in feature space and store cluster centers (Texton dictionary)
 - For test image, filter image with same filter bank to get feature vector for each pixel. Assign each pixel to nearest cluster. Cluster ID = Texton ID.
 - For a given region, compute **texton histograms**
 - Classification: Given new img., compare hist. to other trng. samples and assign to label with most similarity
 - Segmentation: Use texton histograms as a feature
- Perceived Boundaries** - Segmentation by human
 - Idea: Texture gradients indicate boundaries well
 - For each pixel, consider a disk that is split into 2 halves with some orientation
 - Measure texture diff. bet. 2 halves via texton hist.
 - Try all orientations. Orientation with high difference suggests boundaries.

07. Keypoints

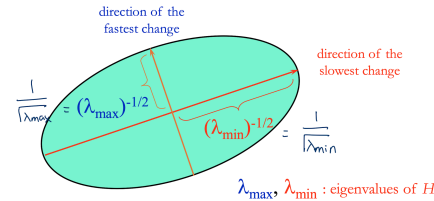
- Motivation: How to stitch 2 images (e.g. Panorama)?
 - Keypoints: Find locations
 - Descriptors: Rep. surrounding regions with math
 - Do the matching
- Good keypoints are **repeatable** and **distinct**
- Harris Corner Detection**
 - Significance: Corners have big changes in all directions when shifting window
 - Given window W shifted by offset (u, v) :

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

- Assuming only small shifts (for Taylor Series Exp.):

$$E(u, v) = Au^2 + 2Buv + Cv^2 = \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

- $A = \sum_{(x, y) \in W} I_x^2$ $B = \sum_{(x, y) \in W} I_x I_y$
- $C = \sum_{(x, y) \in W} I_y^2$
- 2nd Moment Matrix (H)** - Middle matrix
- E is visualized as ellipse, where H controls shape
- Eigenvectors of $H \rightarrow$ Axes orientation
- Eigenvalues of $H \rightarrow$ Axes length



- Eigenvectors** of A are vectors x that: $Ax = \lambda x$
- Eigenvalue (λ)** corresponds to x : $\det(A - \lambda I) = 0$
- Since $A = H$ is 2×2 : $\lambda_{\pm} = \frac{1}{2}((h_{11} + h_{22}) \pm \sqrt{2h_{12}h_{21} + (h_{11} - h_{22})^2})$
- After getting λ s, find x : $(A - \lambda I)x = 0$
- Both λ_{\max} and λ_{\min} are large \rightarrow Corner
- 'Cornerness' Score: $R = \min(\lambda_1, \lambda_2)$ (But getting λ is slow)
- Harris Operator**: $R = \det(H) - \kappa(\text{trace}(H))^2$
 - $\det(H) = AC - B^2 = \lambda_1 \lambda_2$
 - $\text{trace}(H) = A + C = \lambda_1 + \lambda_2$
 - $R > 0 \rightarrow$ Corner, $R < 0 \rightarrow$ Edge, $R \approx 0 \rightarrow$ Flat

 - Compute gradient for each point in image
 - Compute H matrix for each image window and get 'correctness' score
 - Find points with window where $R >$ threshold
 - Take points of local maxima

- Non-Max. Suppression** - Iteratively search for max. values, then zero everything in surrounding window
 - Window size important
 - Adaptive: To prevent uneven distri. of keypoints in areas of higher contrast, pick corners which are both local max. and whose response is greater than all neighboring local max.
- In practice: $H = \sum_{(x, y) \in W} w_{xy} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$ (e.g. Convolve with Gaussian)
- Harris Corner Invariances**
 - Purpose: If img. transf., how repeatable is detection?
 - Equivariance** - Image transformed, and detection location undergoes similar transformation
 - Invariance** - Image transf., but no detection score change
 - Translation: Equivariant and invariant
 - Rotation: Equivariant and invariant
 - Photometric transformation (Assume $I' = aI + b$): Invariant to $b \neq 0$, but not invariant to $a \neq 1$
 - Scaling: Not equivariant and invariant
 - Scale of window can determine if location is keypoint \rightarrow Need to scale up window by image scale
 - Auto. Scale Selection** - When looking for keypoints, try window sizes and find scale that gives local max.

- Laplacian of Gaussian** - Alternative keypoint detector which detects 'blobs' and is scale-sensitive
 - $\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$
 - Practice: Approx. by Difference of Gaussian for speed
 - Idea: Convolution with LoG has highest response when signal has same scale as Gaussian. Built-in scale sensitivity by varying scale σ .
 - Implementation: Fix window and kernel size; rescale img. with Gaussian blurring and downsampling

08. Descriptors

- Goal: Get feature vector surrounding each keypoint and measure similarity between feature vectors for matching
- Desc. should be invariant/equivariant and unique. E.g.:
 - Raw intensity: Good for exact template matching, but sensitive to lighting
 - Image gradient: Invariant to raw intensity (i.e. Lighting), but sensitive to transformations
 - Color histogram: Invariant to scale and rotation, but not sensitive to spatial layout
 - Spatial histogram: Compute color histograms over spatial cells. But not invariant to large rotations.
 - Orientation normalization: Normalize orientation of patch based on dominant image gradient
 - Save orien. angle θ w/ keypoint (e.g. Mean, mode)
 - GIST Descriptor** - Rough spatial distribution of image gradients that is rotation invariant
 - Divide image into 4×4 grid
 - Apply Gabor filters (All dir. edge; N filters)
 - Compute filter response averages for each cells
 - Size of descriptor: $4 \times 4 \times N$
- SIFT** - Keypoint detector and descriptor
 - Detector: Uses multi-scale LoG to get scale invariance, orientation normalization for rotation invariance, and threshold for removing low-contrast and low-curvature keypoints
- SIFT Descriptor**
 - Take 16×16 -pixel window around keypoint. Partition window into 4×4 grid.
 - Compute **gradient orientations and magnitudes** for each pixel. Reweight magnitudes using Gaussian and discard pixels with low magnitude.
 - For each 4×4 -pixel cell, make **histogram with 8 orientation bins**. Shift histogram binning by dominant orien. (i.e. Subtract by dom. orien.) for rotation invariance. Collapse into 1×128 vector.
 - Normalize vector to unit length
 - Invariant to scale, rotation, and lighting
 - Partially invariant to viewpoint (Up to 60°)
 - Quick and efficient
- Feature Matching** - Given feature in I_1 , how to find best match in I_2 ?
 - Define distance function that compares desc.
 - Euclidean distance: $||f_1 - f_2||$ (Can give small distances for incorrect matches)
 - Ratio distance bet. best vs. next-best: $\frac{||f_1 - f_2||}{||f_1 - f_2||}$
 - Nested loop: Find vector with min. distance in I_2
 - Or ratio of best vs. next-best $<$ threshold
 - Evaluation: **ROC curve** by varying threshold
 - Recall vs. 1 - Specificity

- Area Under the Curve (AUC): 1 is the best
- **Recall** - $\frac{TP}{TP+FN}$ **Specificity** - $\frac{TN}{TN+FP}$
- **Precision** - $\frac{TP}{TP+FP}$

09. Homography

- Goal: Stitch images from diff. viewpoints via projection
- When to use: Scene is planar, approx. planar (i.e. Small depth variation), or only camera rotation
- Problem: Given set of matched keypoints $\{p_i, p'_i\}$, get transformation $p' = f(p; H)$ where H are parameters
- Given homography function: Convert to homogeneous coord., multiply by homo. matrix, and convert back

$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}; P' = HP; P' = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} p' = \begin{bmatrix} \frac{x'}{w'} \\ \frac{y'}{w'} \end{bmatrix}$$

- **Direct Linear Transform** - Find best estimate of H
 1. For each matching, create 2×9 matrix A_i
 - $A_i = \begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yy' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix}$
 2. Concatenate into $2n \times 9$ matrix A
 3. Compute SVD of $A = U \sum V^T$
 4. Store vector of smallest singular value $h = v_i$
 5. Reshape to get H
 - Assumptions: Projective model with linear transf.
 - Cons: Sensitive to scaling (i.e. P : High res.; P' : Low res.) \rightarrow Normalize, Outliers \rightarrow Poor est. of H

- **RANSAC** - More robust method for est. homographies
 - Motivation: DLT easily corrupted by outliers

 1. Loop N times
 - (1) Sample randomly num. of pts. required to fit model
 - (2) Solve for model params. using samples
 - (3) Score by frac. of inliers in threshold δ of model
 2. Fit model to samples with most inliers
 1. RANSAC loop
 - (1) Sample 4 matches (H has 8 deg. of freedom)
 - (2) Compute H using DLT
 - (3) Inliers: Get P'' using H and check distance to P'
 - (4) Keep H if largest number of inliers
 2. Using best H with most inliers, recompute using all inliers

- δ : Impacts if inliers are kept (Trial and error)
- $N = \frac{\log(1-p)}{\log(1-(1-e)^s)}$ where p is prob. that ≥ 1 set of samples does not contain outliers, e is prob. that point is outlier, and s is num. of samples per iter.
- Can loop N times or stop early when expected prob. of inliers reached, but both need prob. of outliers
- Integrate RANSAC with feature matching: Compute matches as before, add RANSAC loop and eliminate some matches that do not fit H
- **Warping** - Moves pixels of an image
 - Mapping: Transf. from source to destination via f
 - Resampling: **Splat** if map to bet. pixels, avg. if receive > 1 source

10. Optical Flow

- **Flow** - Displacement of pixels bet. frames (Vector field)
- Assumptions:
 - Color constancy: $I(x, y, t) = C$

- Small motion: $I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$

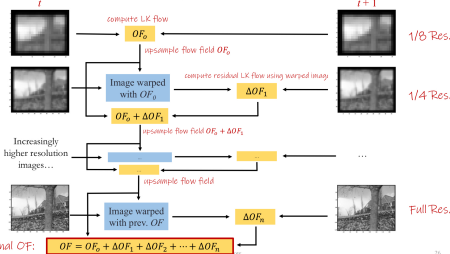
$$I_x u + I_y v + I_t = 0$$
 - $I_t = I(x, y, t + 1) - I(x, y, t)$
 - Problem: 1 equation, 2 unknowns
 - **Lucas-Kanade** - Assumes constant flow in small region
 - Given $m \times n$ patch: $Ax = b; x = (A^T A)^{-1} A^T b$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_{m \times n}) & I_y(p_{m \times n}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_{m \times n}) \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}^{-1} \begin{bmatrix} \sum_{p \in P} I_x I_t \\ \sum_{p \in P} I_y I_t \end{bmatrix}$$

- Requirements: $A^T A$ is invertible $\rightarrow \det(A^T A) = \lambda_1 \lambda_2$ should be big, $A^T A$ is well-conditioned $\rightarrow \frac{\lambda_{\max}}{\lambda_{\min}}$ should be small
 - Produces **sparse flow**: Only for some features
 - Similar to corner detector: Corners good for flow
- Aperture Problem: Given small window over an edge, hard to tell which direction line is moving
 - Solution: Get windows with diff. gradients (Corner)
- Aliasing: Undersampling of frames \rightarrow Nearest match based on intensity is incorrect
 - Similar example: Image motion is large
 - Solution: Reduce res. to reduce apparent movement

Coarse-to-Fine Optical Flow Estimation



- **Horn-Schunck** - Assumes smooth flow field
- Minimization problem: Use gradient descent

$$\min_{u,v} \sum_{i,j} (E_s(i,j) + \lambda E_d(i,j))$$

- $E_s(i,j) = \frac{1}{4}((u_{ij} - u_{i+1,j})^2 + (u_{ij} - u_{i,j+1})^2 + (v_{ij} - v_{i+1,j})^2 + (v_{ij} - v_{i,j+1})^2)$
- $E_d(i,j) = (I_x u_{ij} + I_y v_{ij} + I_t)^2$
- 1. Compute I_x, I_y, I_t and initialize flow $u = v = 0$
- 2. Do until converge: $\hat{u}_{kl} = \bar{u}_{kl} - k I_x; \hat{v}_{kl} = \bar{v}_{kl} - k I_y$

$$k = \frac{I_x \bar{u}_{kl} + I_y \bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} \text{ where } \bar{u}_{kl} \text{ is local avg.}$$

- Choice of λ : When small, maximize smoothness
- Produces **dense flow**: Flow for all pixels
- Good for when image motion is small
- Evaluation: Euclidean distance, Cosine similarity

11. Tracking

- **Tracking** - Estimates parameters (location) of dynamic system (Feature points, target objects) over time

- Use optical flow for tracking? Flow only reliable for small motions, Occlusions and textureless regions (i.e. No corners) get bad est., LK gets sparse fields
- How to match template:
 - Template matching with normalized cross-correlation: Brute force, slow
 - $x_{ij} = \frac{1}{\|F\| \|w_{ij}\|} \sum_{u=-k}^k \sum_{v=-k}^k f_{uv} \cdot p_{i+u, j+v}$
 - Multi-scale template matching: Start search with coarser res., Faster
 - Local refinement based on some initial guess (e.g. Previous frame): Fastest, but sensitive to guess
- Notation:
 - 2D image transformation: $W(x; p)$ where $x = \begin{bmatrix} x \\ y \end{bmatrix}$ and $p = \{p_1, \dots, p_N\}$
 - Brightness constancy assum.: $I(x') = I(W(x; p))$
 - Translation: $W(x; p) = \begin{bmatrix} x + p_1 \\ y + p_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & p_1 \\ 0 & 1 & p_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
 - Affine: $W = \begin{bmatrix} p_1 x + p_2 y + p_3 \\ p_4 x + p_5 y + p_6 \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

- Image alignment problem: Given template $T(x)$ and source img. $I(x)$, solve for transformation parameters p that maps coordinates of img. to template such that: $\min_p \sum_x (I(W(x; p)) - T(x))^2$

Lucas-Kanade Image Alignment

- Assumption: Good initial guess of p and increment Δp from guess p is small \rightarrow Linearize with Taylor Series and solve for Δp

 1. Warp image with initial p : $I(W(x; p))$
 2. Get error image: $T(x) - I(W(x; p))$
 3. Get gradient: $\nabla I(x')$ where x' : Warped img. coord.
 4. Get Jacobian: $\frac{\partial W}{\partial p} = \frac{\partial (W_1, W_2)}{\partial (p_1, \dots, p_N)}$
 5. Get Hessian approx.: $H = \sum_x (\nabla I \frac{\partial W}{\partial p})^T (\nabla I \frac{\partial W}{\partial p})$
 6. $\Delta p = H^{-1} \sum_x (\nabla I \frac{\partial W}{\partial p})^T (T(x) - I(W(x; p)))$
 7. $p = p + \Delta p$

- **KLT Tracker** - Use LK img. alignment to track feature points

- Good features to track: Hessian is well-conditioned matrix \rightarrow Corners!

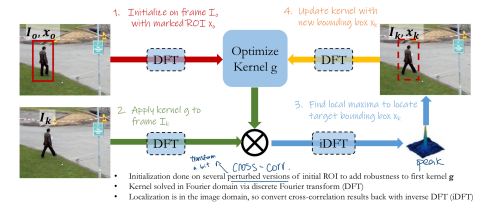
1. Find corners where $\min(\lambda_1, \lambda_2) > \lambda$ for Hessian
2. Loop over corners:
 - (1) Get displacement to next frame using LK alignment
 - (2) Store disp. of each corner, update corner position
 - (3) Optional: Add corners every M frames via step 1
3. Return long trajectories for each corner point

- Challenges: Determining features to track, How to track efficiently, Changing appearance of tracked points, **Drift** - Accumulation of small errors as model updates, Handle addition/removal of tracked points
- Feature tracking vs. Optical flow: Duality

- **Template-based Tracking** - General algorithm to track target objects

1. Mark bounding box around target obj. in 1st frame
2. From target object template, get template descriptor: $q = \{q_1, \dots, q_M\}$
3. In next frame, find similar descriptor in neighborhood
 - Candidate descriptor: $p(y)$ centered at location y
 - Find peak similarity in region: $\max_y \rho(p(y), q)$

4. Update target and descriptor q and repeat
- What features to use? (Naive: Raw intensity; Better: Gradient, hist.)
- Search for candidates (Naive: Brute force image; Better: Use prev. results)
- Finding best candidate (Naive: Simple global max; Better: Use prev. locations)
- Update target's template (Naive: Keep fixed; Better: Update)
- Cross-correlation as Naive Tracker: Crop region of interest as template and apply norm. cross-correlation with template as filter
 - Problem: False local max. when target differs a lot from template \rightarrow Template need to dynamically adapt
- **MOSSE Filter** - Given target x , get unknown filter g to get response y modelled as Gaussian with small σ
 - $g = \arg \min_g \frac{1}{N} \sum_{i=1}^N (g \otimes x_i - y_i)^2 + \lambda \|g\|^2$
 - $g = (X^T X + \lambda I)^{-1} X^T y$
 - N : Total number of target samples
 - X : $N \times n_g$



- Incr. update kernel: Discrete Fourier Transform
 - Use multiplication, instead of X-corr. \rightarrow Faster
 - Kernel remains additive across targets \rightarrow Recently detected target gets higher weight to update kernel
- Issues: Init., Occlusions, Exit frame, Clutter, Drift
- Accuracy: How well does tracker bounding box overlap with ground truth box? $\frac{\text{Area}(B_{gt} \cap B_p)}{\text{Area}(B_{gt} \cup B_p)}$
- Robustness: How many times does tracker fail?

12. Deep Learning

- **Neural Network** - Interconnected perceptrons with biases and weights
- Images as inputs: 1 fully-connected perceptron contains 1 weighted combo. of all input pixels \rightarrow Ignores spatial info. (Neighboring pixels are more correlated) and slow
- Insight: Stationarity (Statistics are similar at diff. locations) \rightarrow Use same weights \rightarrow Same as convolution
- **Convolution** - Apply multiple filters/kernels to image
 - Single filter \rightarrow 1 2D-Feature map/channel
 - **Pooling** - Reduce large distortions \rightarrow More invariance
 - Interleaving conv. with pooling causes later conv. to capture more of the image with same kernel size
 - **Receptive Field** - Set of img. pixels that intermediate output pixel depends on
 - Convolutions after pooling increase receptive field
- **Activation Func.** - Introduces non-linearities
 - $\text{ReLU}(x) = \max(0, x)$
- Principles behind CNN:
 - Local interactions: Spatial locality included and layers will expand from local region to global region
 - Parameter sharing: Stationarity assumption \rightarrow Equivalent representation