1 Intro to CV

1.1 Computer Vision Challenges

- Images/videos comes in a lot of variations in **viewpoints**, **illumination** and **scale**, **intra-class** (e.g. all cars but different brands/models), **motion**, **background clutter**, **occlusion**
- There are a lot of problems when dealing with perception as well (e.g. Objects that are the same size in real life will disappear into horizon)
- Difficult to find algorithms that can generalize and fit all different kinds of variations

1.2 Colors

1.2.1 Converting RGB to Grayscale

- Grayscale Intensity $= W_R \cdot R + W_G \cdot G + W_B \cdot B$, $W_R + W_G + W_B = 1$

1.2.2 Normalized RGB

- $(r,g,b) = \left(\frac{R}{R+G+B}, \frac{G}{R+G+B}, \frac{B}{R+G+B}\right)$
- Sometimes represented as (r,g,I), $I = \frac{R+G+B}{3}$ - Intensity more informative

1.3 HSV Color Space

- **Hue**: "pure" colour (0 to 360)
- **Saturation**: "purity", mixing pure colour (1) with white light (0)
- **Value**: achromatic mixing from 3 Color Images black (0) to white (255)

2 Point Processing and Filtering

2.1 Point Processing

2.1.1 Brightness

- $x_{i,j} = p_{i,j} + b$ - image clips at 0 and 255

2.1.2 Intensity Scaling

- $x_{i,j} = a \cdot p_{i,j}$

2.1.3 Image Normalization (Whitening)

- Removes contrast and constant additive luminance variations
- Mean $\mu = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} p_{ij}}{IJ}$
- Var $\sigma^2 = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} (p_{ij}-\mu)^2}{IJ}$
- $x_{ij} = \frac{p_{ij} - \mu}{\sigma}$

$$H = \begin{cases} \left(\frac{G-B}{V-\min\{R,G,B\}}\right)\cdot 60°, & \text{if } V = R \text{ and } G \geq B; \\ \left(\frac{B-R}{V-\min\{R,G,B\}}+2\right)\cdot 60°, & \text{if } G = V; \\ \left(\frac{R-G}{V-\min\{R,G,B\}}+4\right)\cdot 60°, & \text{if } B = V; \\ \left(\frac{R-B}{V-\min\{R,G,B\}}+5\right)\cdot 60°, & \text{if } V = R \text{ and } G < B \end{cases} \quad H \in [0°, 360°[$$

$$S = \frac{V - \min\{R,G,B\}}{V} \quad S \in [0,1]$$

$$V = \max\{R,G,B\} \quad V \in [0,255]$$

2.1.4 Gamma Mapping

e.g. $x_{i,j} = 255 \cdot \left(\frac{p_{i,j}}{255}\right)^{\gamma}$, non-linear transformation

2.1.5 Histogram Stretching and Equalization

- Histogram stretching: $x = (p - f_1) \times (255/(f_2 - f_1))$, p is the original value, f_1, f_2 is the min/max value of original
- Histogram Equalization: $x_{i,j}$ = max intensity $\times \text{CDF}_{i,j}$

2.1.6 Uses of Histograms

- Segmentation/Thresholding - separating foreground object from background
  - Ideally want the histogram to be bimodal but is highly unlikely in the real world
  - Can achieve automated thresholding using Otsu's Method
  - $T* = \arg\min w_1(T)\cdot\sigma_1^2(T) + w_2(T)\cdot\sigma_2^2(T)$

    $\sigma_1^2(T), \sigma_2^2(T)$ - variance of pixels $\leq$ or $>$ threshold

    $w_1(T), w_2(T)$ - number of pixels $\leq$ or $>$ threshold

2.2 Filters

2.2.1 Noises

Can try to reduce noise using the following methods:

- replacing pixel values with average of neighbors' value
- Weighted Moving Average - give more weight to center pixel
- Gaussian Filter - blurs as well as removes Gaussian noise
- Median Filter - removes spikes (salt and pepper noise) with no new pixel values introduced, may lose edges
  - Strategy is to take all pixel value in a 3x3 grid, sort the values and then take the value of the median

2.2.2 Gaussian Filter

- $f_{uv} = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$, where σ is the scale of gaussian
- Sigma of Gaussian Filters affect the blurring effect more than the size of the kernel

2.2.3 Sharpening Kernel

- -1/9[[1,1,1], [1,-17,1], [1,1,1]]
- Combines a kernel to increase intensity of middle pixel while simultaneously applying a box filter
- do nothing for flat areas but stresses intensity peaks and differences with respect to the surroundings (**edges and noise is emphasized**)

2.2.4 Cross Correlation

- Typically want some strategies to create a padding for images - zero pad, wrap around, copy edge or reflect across edge
- For a kernel with width 2k+1, we need to pad by k pixels

2.3 Template Matching

- Use what you looking for as a kernel and do cross correlation on the image - any parts of the image that "matches" the template will show up as a local maxima
- Problem arises if there are areas where it is primarily white (gives false positive) - solution is to use normalized cross correlation

$$x_{ij} = \frac{1}{|F||w_{ij}|}\sum_{u=-k}^{k}\sum_{v=-k}^{k} f_{uv}\cdot p_{i+u,j+v}$$

* $|\circ|$ = root of sum of squares of all elements, F is the template, w is the window

- Normalized version scales with respect to both filter and corresponding input window
- Template matching is **typically inadequate** for three-dimensional scene analysis due to occlusion, changes in viewing angle and articulation of parts

2.4 Convolution

- To perform convolution, first flip the **kernel** in both direction (up-down, left-right) and apply cross correlation
- Has the same properties as multiplication - commutative, associative, distributes over addition, scalar factors out, identity
- Is **shift invariant** - output depends only on the pattern and not position of neighbors

3 Gradients and Edges

3.1 Why edges?

- Carries lots of information like reflectance changes, textures, appearance, depth discontinuities, object boundaries, shadows etc.
- **Resilient to lighting and color** - useful for recognition
- Give shape and geometry information for 3D understanding

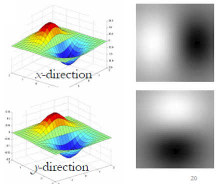
3.2 Gradients

- Edges are sharp discontinuities (first derivative) in intensity
- Gradient is a vector that points in the direction of most rapid change in intensity
  - Gradient direction $\theta = \tan^{-1}\left(\frac{\delta f}{\delta y} / \frac{\delta f}{\delta x}\right)$
  - Edge strength $\|\nabla f\| = \sqrt{\left(\frac{\delta f}{\delta x}\right)^2 + \left(\frac{\delta f}{\delta y}\right)^2}$

3.2.1 Sobel Filter

Horizontal Sobel Filter (gives **vertical** lines): [[1,0,-1], [2,0,-2], [1,0,-1]]

Vertical Sobel Filter (give **horizontal** line): [[1,2,1], [0,0,0], [-1,-2,-1]]



- Other edge detection filters: Scharr (weigh central element more), Prewitt (simplest), Roberts (look at diagonal gradient)
- When looking for edges/gradients, it is **important to blur** it first as **differentiation is very sensitive to noise**
  - note that there is a tradeoff as adding blur will also blur the edge
  - to save on computation, we can convolve a derivative of gaussian (2nd derivative) with the image which requires only 1 convolution

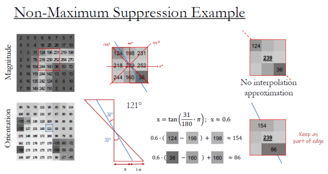
3.2.2 Effect of σ on Derivatives

- Larger σ detects larger-scale edges (good detection) but has poor localization, smaller σ detects finer edges (good localization) but has poor detection

3.2.3 Laplace Filter

- second derivative filter that can also be approximated with finite differences: [[0,1,0], [1-4,1], [0,1,0]] or with diagonals [[1,1,1], [1,-8,1], [1,1,1]]
- Laplacian filter can be used to sharpen images by subtracting the filter from the image (i.e. X - Laplace)
- Can also be combined with Gaussian Filter and applied to images to get edges
  - Laplacian of Gaussian Filtering leads to "zero-crossing" (the edge themselves are black and surrounded by 2 local maxima at either side) which is less convenient as compared to Derivative of Gaussian Filter as it can just be thresholded to get the edge

3.3 Canny Edge Detection



- Aims to convert the thick regions of gradient in the single pixel wide edge
- Steps involved:
  1. Blur image and find gradients using derivative of Gaussian
  2. Find magnitude and orientation of gradient
  3. Non-maximum Suppression - thinning of "ridges" down to single pixel
  4. Hysteresis Thresholding - edge linking

3.3.1 Hysteresis Thresholding

- Apply high and low threshold on NMS image
- Only include the low edges that are linked to the high edges and discard the rest

4 Lines & Hough Transform

4.1 Hough Transform

- Aims to answer the questions on **what is the line** given a number of points, **how many lines** there are and **which points belong to which lines**
- Uses a **voting technique**
  - For each edge point, record a vote for each possible line that passes through that point
  - Look for lines which receive many votes

4.2 Different Forms of Equations for Lines

4.2.1 Slope Intercept Form

$y = mx + b$, where m is the slope and b is the y-intercept

4.2.2 Double Intercept Form

$\frac{x}{a} + \frac{y}{b} = 1$, where a is the x-intercept and b is the y-intercept

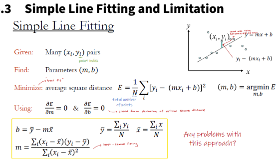
4.2.3 Normal Form

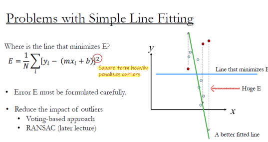
$x\cos\theta + y\sin\theta = \rho$

where θ is the angle between the respective axis to the normal, ρ is the length of the normal

$\rho^2 = 1/(1/a^2 + 1/b^2)$

4.3 Simple Line Fitting and Limitation





4.4 Hough Transform

4.4.1 Image vs Parameter Space

- In image space, y and x are the variables and m and b are the parameters. i.e. $y = mx + b$
- In parameter space, since we already know the x and y coords, they become the parameter instead and we are trying to find the m and b values. i.e. $y - mx = b$

4.4.2 Different Transformation of Hough Transform

- Using the Slope Intercept Form:
  - A line in image space becomes a point in parameter space
  - A point in image space becomes a line in parameter space
- Using the Normal Form:
  - Points in image space becomes sinusoids in parameter space
- For circles:
  - If **radius known**, a point becomes a circle in parameter space
  - If **radius unknown**, a point becomes a cone in parameter space

4.4.3 Line Detection with Hough Transform (Normal Form)

- Problem with the slope intercept form is that the range of accumulator is infinite $(b, m$ can take any values)
- Solve by using normal form since $-\pi \leq \theta \leq \pi$ and $0 \leq \rho \leq \rho_{max}$
- 2 ways to write the same line:
  - Positive rho: $x\cos\theta + y\sin\theta = \rho$
  - Negative rho: $x\cos(\theta+\pi) + y\sin(\theta+\pi) = -\rho$

4.4.4 Hough Transform Algorithm

1. Quantize Parameter Space (θ, ρ)
2. Create Accumulator Array $A(\theta, \rho)$
3. Set $A(\theta, \rho) = 0, \forall \theta, \rho$
4. For each image edge points (x_i, y_i)

   For each element θ

   Solve $\rho = x_i\cos\theta + y_i\sin\theta$

   Increment $A(\theta, \rho) = A(\theta, \rho) + 1$
5. Threshold & find local maxima in accumulator array

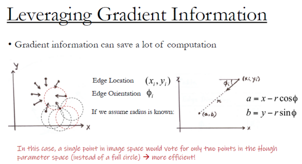
Idea of hough is that each point in image space is transformed into a line in Hough Space. To see whether points lie on the same line, we just have to **find areas where the lines intersect**

4.4.5 Hough Transform and Noise

- Ideally, all the lines will intersect at a point but even with noise, it is still rather easy to discern peaks

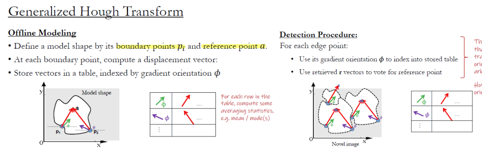
4.5 Parameterizing a Circle

- $(x-a)^2 + (y-b)^2 = r^2$
- If we are using fixed radius, then we can only detect objects of that specific radius (e.g. If r is radius of pennies then can only detect pennies)
- **Robust to occlusions**



- By using gradient information, we are able to reduce the search space from a circle to a line
- 2 points will be $a = x - r\cos\phi, b = y - r\sin\phi$ and $a = x + r\cos\phi, b = y + r\sin\phi$

4.6 Generalized Hough Transform



- To account for orientation and scale changes, we index by matched local patterns instead (visual codewords)

4.7 How to do Voting process

- Run canny on image first to minimize irrelevant tokens
- Choose bin size of accumulator array wisely
  - Too coarse and it would cause different lines to be merged, Too fine and it will miss lines due to noise
- Do "soft voting" for neighbors: $(\theta, \rho) = 0.25*(\theta, \rho-1), 0.5*(\theta\rho), 0.25*(\theta, \rho+1)$
- Limit voting from each token (use direction of edge to reduce votes cast)

4.8 Pros and Cons of Hough Transform

Pros

- All points are processed independently, so can cope with occlusion, gaps
- Some robustness to noise: noise points unlikely to contribute consistently
- Can detect multiple instances of a model in a single pass

Cons

- Search time complexity increases exponentially with the # of model parameters
- Non-target shapes can produce spurious peaks in parameter space
- Quantization: can be tricky to pick a good grid size

5 Image Segmentation

Want to separate image into coherent regions for efficiency of further processing. Segmentation make use of **clustering** but faces 2 key challenges:

- What makes 2 points similar/different - want to represent pixel with **features** (e.g. color using sum of squared difference) and compare using features
- How to compute overall grouping from pairwise similarities
  - **K-Means**: iteratively re-assign points to the nearest cluster center
  - **Mean-shift** clustering: estimate modes of a probability density function

5.1 k-Means Clustering

- Choose k centers as representative and label every pixel based on nearest centers (based on squared different between points and the cluster centers)
- Faces a "chicken and egg" problem where if we know the centers then we can allocate pixels to groups and if we knew group membership we can get center by averaging

Algorithm

1. Given K, randomly initialize the cluster centers, $c_1, \cdots, c_k$
2. Given cluster centers, determine points in each cluster - for each p_i find closest c_j and assign it to that cluster
3. Given points in each cluster, solve for c_j by taking mean of points in cluster j
4. If $c_j >$ threshold, repeat 2 and 3

5.1.1 Math of K-means

- In each iteration, we are minimizing an objective function of $\min_{c,y}\sum_i \|p_i - c_{y_i}\|^2$ (assigning points to nearest center and re-estimating center from taking mean of cluster)
- Eventually converges but likely to **local minimum** - gives a **non-uniform** quantization of image intensities

5.1.2 Feature Selection

- **Intensity** similarities - don't have to be spatially coherent
- **Color** Similarity
- **Intensity + Position** similarity - same colors in 2 different regions considered 2 segments

5.1.3 Pros & Cons

Pros

- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

Cons

- How to set k?
- Sensitive to initial centers and outliers
- Detects spherical clusters
- Assumes means can be computed efficiently and are meaningful

5.2 Superpixeling

Superpixels are a group of pixels that share common characteristics, e.g. pixel intensity. They ar used as inputs to other CV algos since it is a convenient and compact representation.

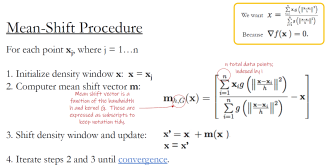
5.2.1 Distance measures

- Color distance: $\sqrt{(r_j - r_i)^2 + (g_j + g_i)^2 + (b_j - b_i)^2}$
- Spatial distance: $\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$
- Composite distance: $\sqrt{d_c^2 + (d_s/s)^2 c^2}$, where s = (# of total pixels / # superpixels)$^{1/2}$

5.3 SLIC Superpixeling Algorithm (Modified k-means)

1. Initialize cluster center on a grid and compute initial superpixel cluster center by moving it to the **lowest gradient position** (for efficiency) in 3x3 neighborhood
2. Assign sample by calculating distance between pixel and closest cluster center
3. Update cluster sample
4. Test for convergence (< Threshold)
5. Post-process - take mean of each superpixel (Optional)

5.4 Mean-Shift Clustering



- Mean shift is heavily influenced by h which is the radius of the kernel - decrease in h will increase # of segments
- Selecting h is by trial and error

5.4.1 Mean Shift Pros and Cons

Pros

- General method of mode-finding
- No prior assumptions on cluster shape
- Only 1 parameter (h) which has a physical meaning
- Finds variable number of modes
- Robust to outliers

Cons

- Output depends on h and selecting h is non-trivial
- Computationally expensive and slow to run
- Scales poorly with feature space dimension

5.4.2 Speedups

- No need to loop through every point - assume all points in a certain radius r will belong to same cluster
- Assign all points within radius c of search path to mode.

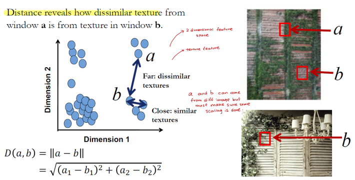
6 Textures

- When we want to find image boundaries, since edges ≠ perceived boundaries, we must rely on textures to associate with physical attribute
- Textures is defined as a **pattern with repeating elements**
- Often, the **same thing can be an object or a texture**, depending on the scale of consideration. (e.g. a singular leaf vs a whole tree of leaves)

6.1 Why Analyze Visual Textures

- **Visual perception**: Indicative of material properties and appearance cues (shapes, boundaries, textures)
- **Computer Vision**: want a feature one step above basic building block of colour, simple filters, and or edges

6.2 How to represent Textures?

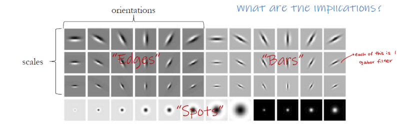


- One way is to use the DoG kernel and apply it to the image, calculate the mean d/dx and d/dy values (using x and y kernel of DoG) of each window and plot it
- Sensitive to window size - too large window will cause signal to be washed out
- Window size is chosen by **scale selection** (trial and error) by looking for window scale where statistic (texture representation) does not change much

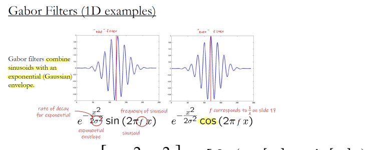
6.3 Generalizing to d filters

- If we apply 2 filters, we get a 2-dimensional feature vector
- By applying d filter, we will have d-dimensional feature vectors - still using euclidean distance to compare "nearness"
- To ensure that we can have a fair comparison between feature points, we can **normalize** them so that scaling does not matter

6.4 Gabor Filter Banks

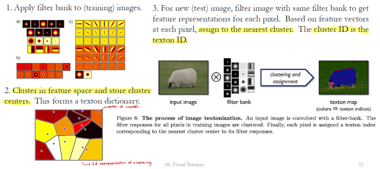


- Typically, a combination of different patterns at various scales and orientations are in a filter bank and each of them are applied to an image to get a n dimensional feature vector
- A easy way to generate the filters is using Gabor Filters

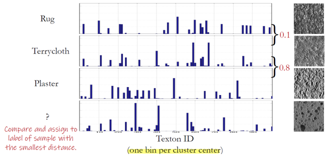


- Gabor 2d: $f_{mn} = \frac{1}{2\pi\sigma^2}\exp\left[-\frac{m^2+n^2}{2\sigma^2}\right]\sin\left[\frac{2\pi(\cos[\omega]m + \sin[\omega]n)}{\lambda} + \phi\right]$, where m, n – coordinates of the kernel (m = n, is circular, otherwise, ellipse), σ - rate of decay, ω - orientation of sinusoid, λ - wavelength of sinusoid, ϕ - phase of sinusoid
- If scale is small compared to the frequency, the Gabor filters ≈ derivative operators

6.5 Textons
- Characterize texture by replacing each pixel with integer representing texture 'type'.



6.5.1 Texton Histograms for Classification



- Each pixel represented with an ID from a texton dictionary.
- Texture is defined by the spatial arrangement of texton IDs, i.e. first order statistic of texton dist
- For a given region, compute a histogram of textons as the representation: vector storing number of occurrences of each texton
- Histogramming helps **reduce the effects of noise** - assumes that similar textures would have similar histograms as a whole

6.5.2 Texton Histograms for Segmentation
- Another use for textons histograms is for segmentation - similar textures having similar histograms are clustered together

6.5.3 Using Textures to Identify Boundaries
- To identify texture boundaries at each location in a scene, we consider a disc, split into two halves by a diameter of a particular orientation. Measure the difference in texture between the two halves by comparing texton histograms and **try all possible orientation**
- **high distance** between histograms, suggests **more likely boundaries**

7 Keypoints & Matching

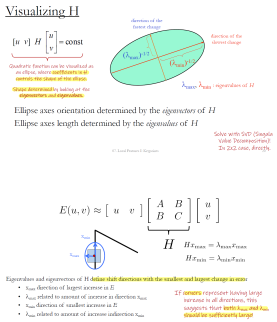
7.1 General Overview - How to combine 2 images
1. Match parts which are the same on both images - Interest point detection, feature descriptors computation, feature matching
2. Align images based on matches (Homography)

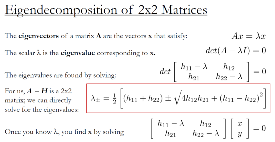
7.2 Characteristics of Good Local Features
- **Repeatable interest points** - (at least some of) same points found in several images despite geometric and photometric transformations (e.g. rotation, scaling, exposure diff)
- **Distinct descriptors** - recognizably different info that allows it to be distinguished
- **Efficient** - number of features « number of pixels; compact
- **Local** - feature occupies a relatively small area of image (robust to clutter and occlusion)

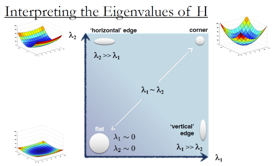
7.2.1 How to look for features
- Want to find regions of image where shifting the window in any direction causes a big change (aka corners)

7.3 Corner Detection
- To detect a corner, we first define a window (W) and shift the window by an offset
- Compare each pixel before and after by summing up the squared differences (SSD) - $E(u,v) = \sum_{(x,y)\in W}[I(x+u, y+v) - I(x,y)]^2$
- If SSD is high then we can consider it a corner - **extremely inefficient** to search for high SSD this way - $O(w*h*m^2*m^2)$, where m is window size

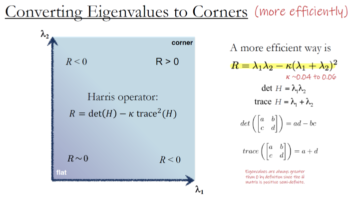
7.3.1 More efficient way of finding corners
- SSD Error $E(u,v) = [u,v]\cdot[[A,B],[B,C]]\cdot[[u],[v]]$
- $A = \sum_{(x,y)\in W} I_x^2$, $B = \sum I_x I_y$, $C = \sum I_y^2$, I_x, I_y - horizontal and vertical gradients







- Special cases of H:
  - Horizontal edge: H = [[0, 0], [0, C]], Vertical edge: H = [[A, 0], [0 ,0]]



- We want to avoid taking $\min(\lambda_1, \lambda_2)$ since calculating eigenvalues requires sqrt which is a computationally expensive operation
- Eigenvalues are always greater than 0 by definition since the H matrix is positive semi-definite
- Runtime: $O(w*h*m^2)$

7.4 Summary of Harris Corner Detection
1. Compute gradient at each point in the image
2. Compute H matrix for each image window and get their cornerness scores
3. Find points whose surrounding window gave large corner response (R> threshold)
4. Take the points of local maxima, i.e., perform nonmaximum suppression

7.4.1 Non-Maximum Suppression
- **Simple** version - search for local maximas then zero-out everything else in window
  - leads to an uneven distribution of interest points in areas of higher contrast
- **Adaptive** version - Pick corners which are both local maxima and whose response is significantly greater (e.g. 10%) than all neighbouring local maxima within some radius r

7.5 Harris Corners Invariances
- **Geometric** Transformation: Rotation and scaling
- **Photometric** Transformation: Intensity changes
- **Equivariance**: if we have two transformed versions of the same image, detection (locations) undergoes a similar transformation
- **Invariance**: image is transformed and detection (score) does not change
- "Detection" refers to corner location as well as probability of being detected as corner
- Harris corners (detected locations) are **equivariant to translation/rotation** and response (R) is **invariant to translation/rotation**
- Harris corner detector is **invariant to additive changes in intensity** (change in brightness) but **not invariant to scaling of intensity** (contrast)
  - To find the optimal scaling, instead of computing f (harris or LoG) for larger and larger window, we can implement using a fixed window size with a **gaussian pyramid** (resize image instead of changing window size)
- Harris corners are **not equivariant to scaling** of image

8 Descriptors

8.1 Definition of Image Feature Descriptor
- Descriptors are **vector representations** that mathematically characterize region in image
- To match images, want to measure the distance between every pair of descriptors
- Descriptors should be **in/equivariant** (shouldn't change under geometric/photometric transformation) and **discriminative** (unique for each point)

8.2 Descriptors Variants
- Pure intensity values - good if geometry/appearance unchanged (template matching)
- Image Gradients - invariant to absolute intensity values
- Color histogram - invariant to changes in scale and rotation
- Spatial histogram - some invariance to deformation but not invariant to large rotations
- Rotation invariant descriptors
  - Harris Corner response measure - depends only on eigenvalues
  - Local orientation - Dominant gradient direction for image patch and rotate patch according to that angle (**canonical orientation**)
    - To find dominant gradient, we can either average the orientations derived from gradients in region around keypoint (may overly smooth out signal and get wrong gradient direction) or use mode

8.3 Multi-Scale Oriented Patches (MOPS)
- 40x40 window around keypoint, subsample every 5th pixel (absorb localization error), rotate to horizontal (gives rotation invariance), normalize intensity value (robust to photometric variations), wavelet transform to 8x8 patch to get 64-dimensional descriptor

8.3.1 GIST Descriptor
- Divide image (patch) into 4x4 cell, apply Gabor Filter, compute filter response averages of each cell, size of descriptor is 4x4xN where N is size of filter bank
- GIST descriptor give a rough spatial distribution of the image gradients

8.4 Scale Invariant Feature Transform (SIFT) Descriptors
- Originally describes both a detector and descriptor
- Descriptor is invariant to scale and rotation, can handle changes in viewpoint (up to 60° of plane rotation), can handle significant changes in illumination, quick and efficient

8.4.1 SIFT Descriptor Algorithm
1. Take 16x16 window around detected keypoint at image at scale matching to keypoint. Partition window into a 4x4 grid of cells (gives some sensitivity to spatial layout)
2. Compute gradient orientations and magnitudes for each pixel; reweight magnitudes according to a Gaussian centered on keypoint and discard pixels with low magnitude
3. Of the remaining edge orientations, create a histogram with 8 orientation bins for each cell. To be rotation invariant, "shift" histogram binning by the dominant orientation. Collapse into vector (16 x 8 = 128 dims).
4. Normalize vector to unit length, clamp values based on threshold, re-normalize again for final descriptor

8.5 Feature Matching

8.5.1 Feature Distance
- Simplest approach: L2 distance (euclidean) - $\|f_1 - f_2\|$ (sum of squared difference)
  - Possibility of giving small distances for ambiguous matches
- Better approach - ratio distance: $\|f_1 - f_2\|/\|f_1 - f_2'\|$, f_2' is the 2nd best match

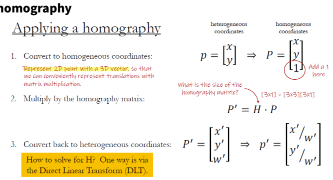
8.5.2 Evaluating Feature Matches
- Precision: TP / (TP + FP), Recall: TP / (TP + FN), Specificity: TN / (TN + FP)
- If threshold ↑, TP and FP ↑

9 Homography

To stitch images from different viewpoints, it is not enough to use translation only stitching. Homography is applied to **project images onto the same plane**

9.1 Image Projections
Translation, Euclidean (Rotation), Similarity (Scaling), Affine (diff x-y scaling but still parallel aka parallelogram), Projective (Homography, resulting image look like a rhombus)

9.2 Applying a homography



9.3 Solving for H using DLT
Given p_i, p_i' solve for H such that $P_i' = H \cdot P_i \forall i$
1. For each correspondence, create 2x9 matrix A_i = $[[-x, -y, -1, 0, 0, 0, xx', yx', x'], [0, 0, 0, -x, -y, -1, xy', yy', y']]$
2. Concatenate into single $2n \times 9$ matrix A
3. Compute SVD of $A = U\Sigma V^T$
4. Store singular vector of smallest singular value $h = v_{\hat{i}}$
5. Reshape to get H (3x3 vector)

9.3.1 Drawbacks of DLT
- DLT uses linear least squares estimation which is only applicable when actual transformation between two images is linear
- Sensitive to scaling in pixel space - solved by normalization (compulsory for DLT)
- DLT does not deal well with outliers - solved using RANSAC

9.4 When are Homographies Applicable
- Scene is planar or approximately planar (very far away or has small depth variation)
- scene is captured under camera rotation only

9.5 RANSAC
- DLT find average transform using L2 distance, easily corrupted by bad correspondences
- RANSAC aims to solve that by having **random sample consensus** for a set of noisy observations as a 2 stage process - classify inlier vs outlier, fit model to only inliers

9.5.1 RANSAC Loop
1. Randomly get (min 4) correspondence - take as little as possible to minimize chance of picking outlier (4 because H matrix has 8 degree of freedom, each point gives 2 equations)
2. Compute H using DLT and count inliers, keep H if largest number of inliers

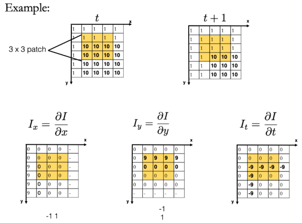
For best H with most inliers, recompute using all inliers

9.5.2 RANSAC Parameters
- ∂ - distance threshold to consider as inlier (obtained using trial and error)
- N - number of iterations of RANSAC loop (can be solved using ratio of outlier vs inlier)
  - N = $\dfrac{\log(1-p)}{\log(1-(1-e)^S)}$, where e is the probability that a point is an outlier, p is the probability that at least one set of points sampled does not contain any outliers, s is the number of points we draw per iteration (2 for a line, 3 for quadratic)

9.6 Image Warping
- Based on the homography, we can warp image into new plane, however discretization causes problems
  - certain destination pixels are not covered
  - many pixels might map to the same destination - solved by "distributing" (averaging) intensity over neighbouring pixels (splatting)

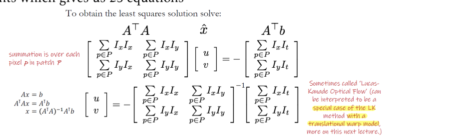
10 Optical Flow
- Optical flow is an instantaneous velocity measurement for direction and speed of image data across visual field
- Key assumptions: Color constancy (allows for pixel to pixel comparison), small motion (allows for linearization of brightness constancy constraint)

10.1 Brightness Constancy Equation
Example:



- $\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$
- $I_x u + I_y v + I_t = 0$
  I_x, I_y is the x and y gradients, u, v are the flow velocities, I_t is the temporal gradient
- I_x, I_y are computed using sobel/scharr filters, I_t is computed using frame $t+1-t$
- Currently, solving the equation will give us solutions that lie on a line (infinitely many solutions), need more constraints to limit solution space

10.2 Lucas-Kanade Flow
- Constant flow assumption - assumes that surrounding patch in 5x5 area has same displacements which gives us 25 equations



- For $A^T A\hat{x} = A^T b$ to be solvable:
  - $A^T A$ should be invertible, determinant of $A^T A$ should not be too small, λ_1, λ_2 should not be too small, λ_1/λ_2 should not be too large ($\lambda_1 >> \lambda_2$)
- Lucas-Kanade **works best on corners** where λ_1, λ_2 are big

10.3 Aperture Problem
- If a motion sensor has a finite receptive field, it makes a homogeneous contour seem locally ambiguous (we can only find out the u or v components)
- Want patches with different gradients to avoid aperture problem (aka corners)

10.4 Gaussian Pyramid
- When movements are too large (> 1px), we face a problem of aliasing where the nearest match is a wrong correspondence
- To fix this, we want to downsample the image to get "smaller" motions (solved using LK) and then iteratively apply this motion to the original image and upsample
- Ideally this brings the original image to a point where it has only a small motion relative to the next reference frame

10.5 Horn-Schunk Optical Flow
- Enforce brightness constancy - for every pixel: $\min_{u,v}[I_x u_{i,j} + I_y v_{i,j} + I_t]^2$
- Enforce smooth flow field - $\min_u (u_{i,j} - u_{i+1,j})^2$, patch don't share same (u,v)
- Key idea - $\min_{u,v}\sum_{i,j} E_s(i,j) + \lambda E_d(i,j)$, E_s is the smoothness and E_d is the brightness constancy (uses gradient descent algo)

10.5.1 Horn-Schunk Algorithm
1. Precompute image gradients I_y, I_x and temporal gradient I_t
2. Initialize flow field ($u = 0, v = 0$)
3. While not converged → compute flow field updates

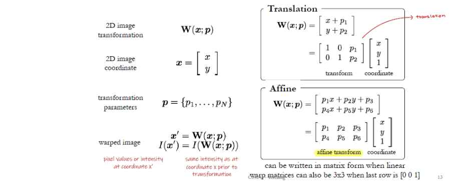
$$\hat{u}_{kl} = \bar{u}_{kl} - \frac{I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_x,\ \hat{v}_{kl} = \bar{v}_{kl} - \frac{I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_y$$

10.6 Lucas-Kanade vs Horn-Schunk
Lucas-Kanade
- feature-based method that extracts visual features and tracks them over multiple frames
- Generates only sparse motion fields, but is more robust at tracking
- Suitable when image motion is large (10s of pixels)

Horn-Schunk
- Directly method that recovers image motions at every pixel based only on spatio-temporal image brightness variations
- Dense motion fields, but sensitive to appearance variations
- Suitable for video and when image motion is small

11 Tracking

11.1 Methods of finding template in image
- Template matching - slow, combinatory, finds global solution
- Multi-scale template matching - scale **image** and match template, faster, local optima
- Local refinement based on some initial guess - fastest, local optima



11.2 Lucas-Kanade Alignment
- Warp image - $I(\mathbf{W}(x;p))$
- Compute error image - $[T(x) - I(\mathbf{W}(x;p))]$, Compute gradient - $\nabla I(x')$
- Evaluate Jacobian - $\partial W/\partial p$
- Compute Hessian Approximation - $H = \sum_x [\nabla I \cdot \partial W/\partial p]^T [\nabla I \cdot \partial W/\partial p]$
- Compute - $\Delta p = H^{-1}\sum_x[\nabla I \cdot \partial W/\partial p]^T[T(x) - I(W(x;p))]$
- Update parameters - $p \leftarrow p + \Delta p$

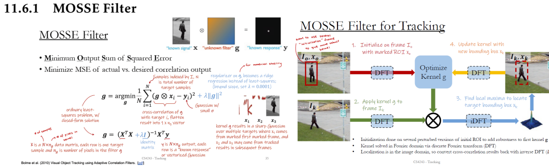
11.3 How to choose good features for tracking
- Want to avoid smooth regions and edges - corners chosen as good features

11.4 KLT Algorithm for Feature Tracking
- Find corners satisfying $\min(\lambda_1, \lambda_2 > \lambda)$ for Hessian
- Loop over corners → compute displacement to next frame using Lucas-Kanade, store displacement and update corner positions, (optional) add more corners every M frames
- Return long trajectories for each corner

11.5 Challenges of Feature-Based Tracking
- Figuring out which features can be tracked
- Efficiency: How to track them efficiently
- Accuracy: Changing appearance of some points e.g. rotations, movement into shadows, drift - accumulation of small errors as appearance model updates
- Appearance / disappearance of points

11.6 Tracking with Template Matching
- Initialize bounding box of object to be tracked manually
- Compute template descriptor for target
- Search for similar descriptor in neighborhood of next frame - look for maxima
- Update target and descriptor
- Template-based tracking algorithm differ in the way they:
  - Represent candidates: gradient features, histogramming, deep features
  - Search for candidates: limit search space based on previous results
  - Solve the mode-finding problem: leverage previous locations to help with local max
  - Update target's template: update as the track progresses
- Evaluation Measure: $|B_{gt} \cap B_p|/|B_{gt} \cup B_p|$

11.6.1 MOSSE Filter



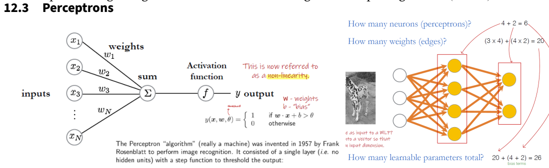
11.7 Tracking Issues
- Initialization typically done manually but background subtraction detection is also used
- Catastrophic errors: Occlusion and clutter, exit of frame, multiple objects
- Gradual errors: drifting due to accumulation of small errors over time

12 Deep Learning

12.1 Data-Driven Image Classification
Collect db of img w/ labels → use ML to train img classifier → evaluate classifier on test img

12.2 AI vs ML vs Deep Learning
- **AI** - imparting cognitive abilities to machine, **ML** - algo and models to perform tasks based on patterns/inference instead of specific instructions, **DL** - branch of ML using neural nets (uses cascade of multiple layers of nonlinear processing units for feature extraction/transformation)
- Deep Learning = Big Data + labels + Known Algos + Computing Power (GPUs)

12.3 Perceptrons



12.4 Stationarity
- To reduce number of parameters in a neural network, we make use of **stationarity** which states that statistics is similar at different locations (e.g. all edges will have same characteristics no matter where they are)
- We now take each "local" perceptron as a filter/kernel that all has the **same learned weight** w and then do convolution over the entire image
  - Param is **independent of img size** and is calculated by # of filters × kernel size
- Output = 0 if $w \cdot x + b \leq 0$ or 1 otherwise
  - Output if using the stationarity property will be a 2d vector since each output is linked to an area in the original image
- Locally Linear + Stationarity = Convolution
  - extent of local connectivity determines kernel size
  - Single filter → single feature map, multiple filters → multiple feature maps
  - Each feature map is a channel of output
  - **Param sharing** built in, leads to equivariant representation - output changes w/ input

12.5 Pooling
- Pooling makes large distortions become smaller to give more invariance
- Max pooling: take max value in a $n \times n$ area

12.6 Convolution-pooling-convolution
- Interleaving convolutions and pooling causes later convolutions to capture a larger fraction of the image with the same kernel size
- Set of image pixels that an intermediate output pixel depends on = receptive field
  - assume that in a neural net layer we care about "local neighborhoods"
  - Composition of layers will expand from local to global
- Convolutions after pooling increase the receptive field
- e.g. of CNN flow → Conv -> ReLU -> Conv -> ReLU -> Pool -> ⋯