CS4243

AY23/24 Sem 2

github.com/jasonqiu212

01. Introduction

- $f(x,y) = i(x,y)r(x,y)$ where f is intensity, i is illumination, and r is reflectance
- **Exposure time** - Time for incident light to reach sensor
- **Storage** - $b = MNK$ where img. has size $M \times N$ and K-bit depth
- Color to greyscale: $I = W_R R + W_G G + W_B B$ where $\sum_{i \in [R,G,B]} W_i = 1$
  - Common weights: $(.299, .587, .114)$
- **Norm. RGB** - $(r,g,b) = (\frac{R}{R+G+B}, \frac{G}{R+G+B}, \frac{B}{R+G+B})$
- $(R,G,B) \leftrightarrow (r,g,b) \leftrightarrow (r,g,I)$ where $I = \frac{R+G+B}{3}$
- **HSV Color Space**
  - Hue: Pure color (0 to 360)
  - Saturation: Mix pure color (1) with white light (0)
  - Value: Mix from black (0) to white (255)

02. Filtering

- **Point processing** - $x_{ij} = f(p_{ij})$
  - **Brightness** - $x_{ij} = p_{ij} + b$ (Clipping behavior)
  - **Intensity scaling** - $x_{ij} = ap_{ij}$ (Increased/decreased contrast)
  - **Normalization (Whitening)** - $x_{ij} = \frac{p_{ij} - \mu}{\sigma}$ $\sigma^2 = \frac{\sum_{i=0}^{I}\sum_{j=0}^{J}(p_{ij}-\mu)^2}{IJ}$
  - **Gamma** - $x_{ij} = 255(\frac{p_{ij}}{255})^{\gamma}$ where $\gamma > 0$
  - **Intensity Histogram** - No data on location
    - **Stretching** - $x = (p - f_{\text{min}})(\frac{255}{f_{\text{max}} - f_{\text{min}}})$
    - **Equalization** - Turn cumulative distribution linear
      1. Get histogram: $h_k = \sum_{i=0}^{I}\sum_{j=0}^{J} \delta$ where $\delta = 1$ if $p_{ij} - k = 0$ and 0 otherwise
      2. Estimate CDF: $c_k = \frac{\sum_{l=1}^{k} h_l}{IJ}$
      3. Map $p_{ij} = k$ to new bin x_{ij}: $x_{ij} = 255\text{CDF}(k)$
      - Pros: More contrast than normalization, less prone to outliers than stretching
      - Cons: More expensive
    - Good for foreground and background separation: Bi-modal histogram can be thresholded
    - **Otsu's Method** - Automated thresholding
      - $T^* = \min_{T \in [0,255]}(w_1(T)\sigma_1^2(T) + w_2(T)\sigma_2^2(T))$ where T^* is the optimal threshold that min. sum of weighted variances of object and background
      - $\sigma_1^2(T)$ and $\sigma_2^2(T)$: Variance of pixels less than or equal to and greater than threshold respectively
      - $w_1(T)$ and $w_2(T)$: Number of pixels less than or equal to and greater than threshold
- **Correlation Filtering** - Window (Moving average)
  - Motivation: Reduce noise
    - Impulse noise: Random white occurrences of pixels
    - Salt and pepper noise: Rand. white and black pixels
    - Gaussian noise: Variations in intensity from Gaussian distribution $p_{ij} = \hat{p}_{ij} + \eta$ where $\eta \sim N(\mu_n, \sigma_n)$
  - $x_{ij} = \sum_{u=-k}^{k}\sum_{v=-k}^{k} f_{uv} \cdot p_{i+u,j+v}$ where f is a **kernel** of weights
  - Notation: $X = P \otimes F$
  - **Box Filter** - Blurs image

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

  - Given kernel of width $2k+1$, add padding: $\lfloor \frac{2k+1}{2} \rfloor$
    - Filling methods: Zero-padding, wrap around, copy edge, reflect
  - **Gaussian Filter** - Nearest neighboring pixels have more weight
    - $f_{uv} = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

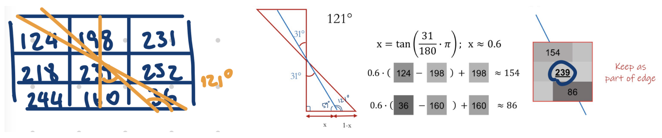
    - Size of kernel: At some threshold, kernel will have lots of 0s, which is not useful
    - Variance: Determines smoothing effect
- **Template Matching** - Set template as kernel and match occurs at local max.
  - **Normalized Cross-Correlation** - Normalizes using filter and input window
    - Motivation: Non-normalized is dominated by original image pixels
  - $x_{ij} = \frac{1}{|F||w_{ij}|}\sum_{u=-k}^{k}\sum_{v=-k}^{k} f_{uv} \cdot p_{i+u,j+v}$ where $|\cdot|$ is magnitude of kernel (i.e. Square root of sum of squares)
- **Convolution Filtering** - Flip kernel in both directions, then do cross-correlation
  - $x_{ij} = \sum_{u=-k}^{k}\sum_{v=-k}^{k} f_{uv} \cdot p_{i-u,j-v}$
  - Notation: $X = F * P$
  - Convolution has nicer properties than correlation
  - Properties: Commutative, Associative, Distributive over addition, Scalar factor, Identity
- **Non-Linear Filters** - Filters that perform non-linear operations (e.g. Median, Min., Max.)
  - Correlation and convolution are both linear operations
  - **Median Filtering**
    - Removes spikes: Good for removing impulse and salt and pepper noise
    - No new pixel values introduced
    - Edge preserving

03. Gradients

- **Gradient** - Vector at pixel pointing at most rapid change in intensity
  - $\nabla f = \langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \rangle$
  - In images, der. is finite: $f'(x) = \frac{f(x+1) - f(x-1)}{2}$
  - $\theta = \tan^{-1}(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x})$
  - $||\nabla f|| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$
- **Sobel Filter**
  - Horizontal sobel filter (S_x): Detects vertical lines
  - Vertical sobel filter (S_y): Detects horizontal lines

$$\boldsymbol{S_x} = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline\end{array} \qquad \boldsymbol{S_y} = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline\end{array}$$

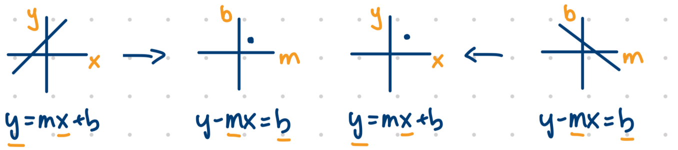
  - Differentiation is sensitive to noise: Need to blur first
    - Trade-off: Blur removes noise, but blurs the edge
    - **Derivative of Gaussian** - Blur + Derivative merged into 1 filter; $\frac{\partial}{\partial x}(h * f) = (\frac{\partial}{\partial x}h) * f$
- **Laplace Filter** - 2nd derivative filter
  - $f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$
  - **Laplacian of Gaussian (LoG) Filter** - Combine Laplace filter with Gaussian
    - Produces zero-crossings at edge
- **Canny Edge Detector**
  1. Filter image with derivative of Gaussian
  2. Find magnitude and orientation of gradient
  3. Thin wide ridges down to single pixel width via non-maximum suppression
  4. Perform thresholding (hysteresis) and linking
- **Non-maximum Suppression** - For each pixel, check if it's local maximum along gradient direction. If yes, then keep. Else, 0.
  - Interpolation:



  - Alternative: Approx. using closest pixel value
- **Hysterisis Thresholding** - High threshold to start edge curves and low threshold to continue growing
  - Motivation: Single high threshold can miss out some parts of the edge

04. Lines

- Goal: Which points belong to which line?
- $y = mx + b \qquad \frac{x}{a} + \frac{y}{b} = 1$
- $x \cos\theta + y \sin\theta = \rho$
- **Line fitting** - $E = \frac{1}{N} \sum_i (y_i - (mx_i + b))^2$
  - Goal: Find parameters m and b that min. E
  - Using gradient descent: $b = \bar{y} - m\bar{x} \qquad m = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$ where $\bar{y} = \frac{\sum_i y_i}{N}$ and $\bar{x} = \frac{\sum_i x_i}{N}$
  - Cons: Choice of error function, outliers
- **Hough Transform** - Paramterize shape and vote



1. Initialize accumulator array $A(\theta, \rho) = 0$
2. For each image edge point (x_i, y_i):
   1. For each θ:
      1. Solve for $\rho = x_i \cos\theta + y_i \sin\theta$
      2. $A(\theta, \rho) = A(\theta, \rho) + 1$
3. Find maximum in $A(\theta, \rho)$
4. Detected lines are given by $\rho^* = x \cos\theta^* + y \sin\theta^*$

- **Circle** - $(x - a)^2 + (y - b)^2 = r^2$
  - If r is unknown, need to solve for 3 parameters (i.e. $A(a, b) \to A(a, b, r)$) and a point becomes a cone in 3D parameter space
  - Optimization: Use gradient
    - Use edge orientation ϕ_i to vote for 2 points, rather than whole circle
- **Generalized Hough Transform** - Arbitrary shape

1. Given shape with boundary points p_i and reference point a
2. For each p_i, get displacement vector from a
3. Store in table with key ϕ_i and value (Displacement vectors)
4. For each edge pt., use ϕ_i to get vectors to vote for a

- Application: Index with local pattern, instead of gradients
- Tips: Soft voting, convert to edge image
- Pros: Food with noise and occlusion
- Cons: Complexity with num. of parameters, grid size

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

1. Randomly initialize the cluster centers $c_1, \ldots, c_K$
2. For each point p_i, find the closest c_j to put p_i in
3. Set c_j to be mean of points in cluster j
4. 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
  - $D = \sqrt{(\frac{d_c}{d_{cm}})^2 + (\frac{d_s}{d_{sm}})^2} = \sqrt{d_c^2 + (\frac{d_s}{s})^2 c^2}$

1. 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.
2. For each cluster center, check distance to all pixels within $2s \times 2s$ neighborhood. Assign pixels to closest checked center.
3. Update cluster centers using mean and repeat if not converged (Same as k-Means)
4. 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

1. For each data point:
   1. Define window around and get centroid
   2. Shift window to centroid
   3. 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 $\to$ 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