$f(x,y) = i(x,y)\, r(x,y)$

$i(x,y)$: source illumination incident $0 \le i(x,y) < \infty$

$r(x,y)$: amount of illumination reflected by objects. $0 \le r(x,y) \le 1$ total absorption to total reflectance

Demosaicing Bayer Patterns

$[R,G,B]_B = [f(x+1,y+1), f(x+1,y), f(x,y)]$

$[R,G,B]_{GB} = [f(x,y+1), f(x,y), f(x-1,y)]$

$[R,G,B]_{GR} = [f(x+1,y), f(x,y), f(x,y-1)]$

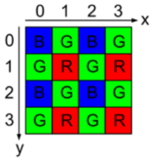
$[R,G,B]_R = [f(x,y), f(x-1,y), f(x-1,y-1)]$

Greyscale

$I = W_1R + W_GG + W_BB,\ \sum W_x = 1$

Normalized RGB:

r/(R+G+B), g/(R+G+B), b/(R+G+B), I=(R+G+B)/3

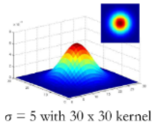


HSV

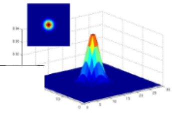
$$H = \begin{cases} \frac{G-B}{V-\min\{R,G,B\}} \cdot 60^\circ, & \text{if } V = R \text{ and } G \ge B; \\ \left(\frac{B-R}{V-\min\{R,G,B\}} + 2\right) \cdot 60^\circ, & \text{if } G = V; \\ \left(\frac{R-G}{V-\min\{R,G,B\}} + 4\right) \cdot 60^\circ, & \text{if } B = V; \\ \left(\frac{R-B}{V-\min\{R,G,B\}} + 5\right) \cdot 60^\circ, & \text{if } V = R \text{ and } G < B \end{cases}$$

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

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



σ = 5 with 30 x 30 kernel

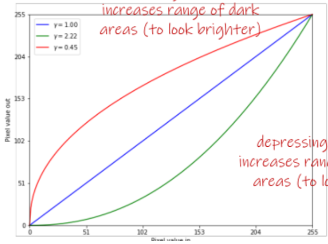


with 30 x 30 kernel

Gamma Mapping

$$x_{ij} = 255 \cdot \left(\frac{p_{ij}}{255}\right)^\gamma, \quad \gamma > 0$$

increasing mid-levels
increases range of dark
areas (to look brighter)



depressing mid-levels
increases range for bright
areas (to look darker)

Image Normalization

$$\text{mean} \quad \mu = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} p_{ij}}{IJ}$$

$$\text{variance} \quad \sigma^2 = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} (p_{ij}-\mu)^2}{IJ}$$

resulting image pixels are zero-mean, unit variance

$$x_{ij} = \frac{p_{ij} - \mu}{\sigma}$$

→if gde

Hist equalization

x=cdf(x)*cap eg: 0.3*255

Change intensity

Hist Stretching:

x=(p-min)*cap/(max-min)

eg: (p-50)*255/(200-50)

Stretch min and max

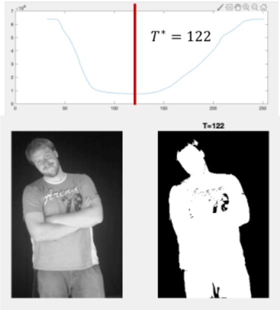
Gaussian Noise

Obs = Ideal + noise

$$p_{ij} = \hat{p}_{ij} + \eta$$

$$\eta \sim \mathcal{N}(\mu_n, \sigma_n)$$

Otsu's Method – Automated Thresholding



$$T^* = \underset{T}{\operatorname{argmin}}\ w_1(T)\cdot\sigma_1^2(T) + w_2(T)\cdot\sigma_2^2(T)$$

- optimal threshold T^* minimizes the sum of weighted variances of object and background.
- rationale:
  - correct threshold produces two narrow modes
  - incorrect threshold produces (at least one) wide mode
  - width of the mode is measured by the variance σ^2
  - weighting by number of pixels in each mode
- $\sigma_1^2(T), \sigma_2^2(T)$ - variance of pixels less than or equal to and greater than threshold respectively
- $w_1(T), w_2(T)$ - number of pixels less than or equal to and greater than threshold

we want to minimise σ for each mode

02. Point Processing & Filtering

Norm-CC

Conv/CC matches white

Norm-CC matches template

Scale to sum up to 1

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

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

Sharpening:

stresses intensity peaks and differences with respect to the surroundings

Gaussian Blur

$$\frac{1}{16}\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

Box Blur (Mean Filter)

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

Sharpen

$$\begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$$

Outline

$$\begin{vmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{vmatrix}$$

Emboss

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

Median filter (denoise/delete pixels)

Hori Sobel (Hori Grad=vert line)

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

Vert Sobel (Vert grad=hori line - -)

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

Identity

$$\begin{vmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix}$$

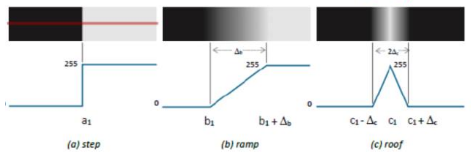
Shift Right

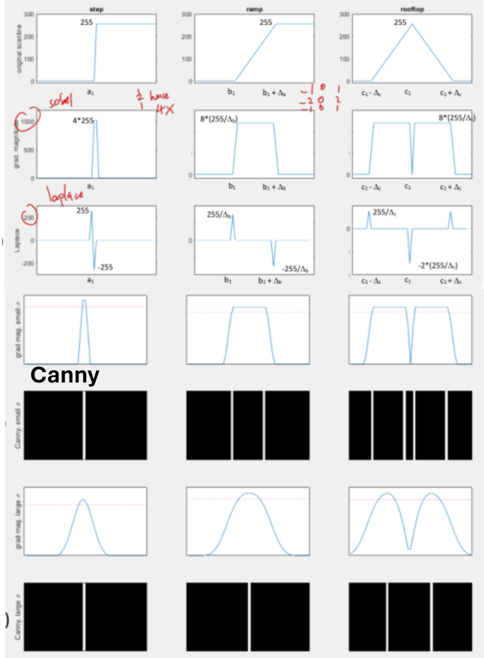
$$\begin{vmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{vmatrix}$$

Laplace (transitions)

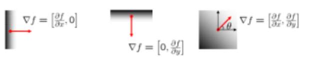
$$\begin{vmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{vmatrix}$$

(2 edges)





$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$ The gradient is a vector that points in the direction of most rapid change in intensity.

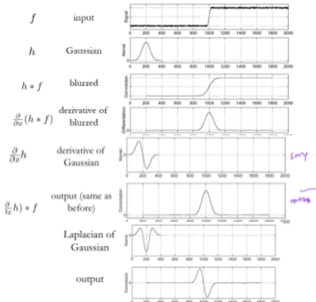


The **gradient direction** (orientation of edge normal) is given by:

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

The **edge strength** is given by the gradient magnitude

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



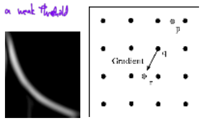
Canny Edge Detector

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) & linking:
   - Mark pixels that pass high threshold as edge
   - Iteratively check and relabel pixels clearing low-threshold as edge if they are linked to an existing edge pixel

min value is 0 because set to 0 in NMS but gradient operation can be above 255

Values are {0, [threshold, ∞)}

you need to tune what is a weak threshold and a strong threshold



Hysteresis thresholding:

use a high threshold to start edge curves, and a low threshold to continue growing them. Iteratively mark adj pixels

1. Find image gradients
   - G : gradient magnitude image (continuous range of values)
2. Apply Non-Maximum Suppression to G
   - G_{NMS} : single pixel gradient image after NMS (continuous range of values)
3. Apply double thresholding
   - 3.1. Apply high threshold H to G_{NMS} to get E_H (binary output; pixels with values above or equal to threshold set to 1, pixels with values below set to 0)
   - 3.2. Apply low threshold L to G_{NMS} to get E_L (binary output)
   - 3.3. $E_w = E_L - E_H$ are weak pixels that passed the low threshold but failed the high threshold (binary output)
4. Linking
   - 4.1. $E_{out} = E_H$ is the initial output edge image (binary)
   - 4.2. E_w' = pixels of E_w which are linked or connected to current E_{out}
   - 4.3. Update $E_{out} = E_{out} + E_w'$; Remove E_w' from E_w by setting to 0.
   - 4.4. Repeat 4.2, 4.3 while there are new pixels added to E_{out} or some other stopping condition

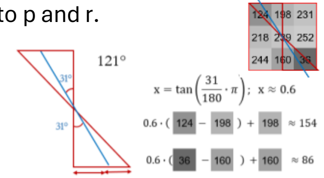
Non-Maximum Suppression (NMS)

For each pixel, check if it is a local max along the gradient direction.

If so, keep as edge, if not, discard (set to 0).

May require checking interpolated values (e.g. pixel locations p and r)

Alternative is to approximate with the closest pixel locations to p and r.



$x = \tan\left(\frac{31}{180}\cdot\pi\right);\ x \approx 0.6$

$0.6\cdot(124 - 198) + 198 \approx 154$

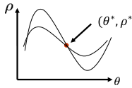
$0.6\cdot(36 - 160) + 160 \approx 86$

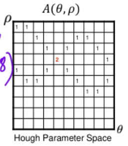
Normal Form

Line Detection with Hough Transform

Algorithm:

1. Quantize Parameter Space (θ,ρ)
2. Create Accumulator Array $A(\theta,\rho)$
3. Set $A(\theta,\rho) = 0 \quad \forall\theta,\rho$
4. For each image edge point (x_i, y_i)
   For each element θ ← if you know gradient ab from Canny, then fewer points to check (slide 28)
   Solve $\rho = x_i\cos\theta + y_i\sin\theta$
   Increment $A(\theta,\rho) = A(\theta,\rho) + 1$
5. Threshold, find local maxima in $A(\theta,\rho)$
6. Detected line(s) given by $\rho^* = x\cos\theta^* + y\sin\theta^*$

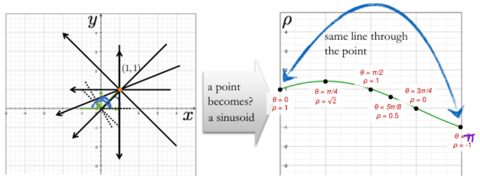




Hough Parameter Space

variables: $y = mx + b$ parameters

parameters: $x\cos\theta + y\sin\theta = \rho$ variables



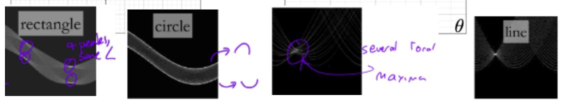
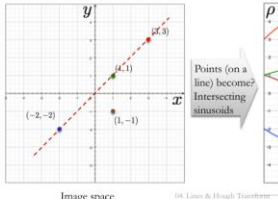


Image and (Normal) Parameter Space

$y = mx + b$

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



There are two ways to write the same line: cos we limit π and −π

Positive rho version:

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

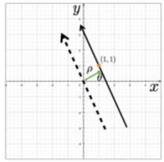
Recall:

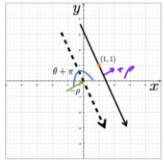
$\sin(\theta) = -\sin(\theta + \pi)$

$\cos(\theta) = -\cos(\theta + \pi)$

Negative rho version:

$x\cos(\theta + \pi) + y\sin(\theta + \pi) = -\rho$





What if Radius is Unknown?

$(x-a)^2 + (y-b)^2 = r^2$ → $(x-a)^2 + (y-b)^2 = r^2$

Radius is now a third parameter.

When considering the parameter space, this adds a third variable → makes the Hough space 3D.

- Augment accumulator array from $A(a,b)$ to $A(a,b,r)$
- In 2D parameter space, a point in image space corresponds to a circle (of radius r)
- In 3D parameter space, a point projected from image space also gets more complicated → cone

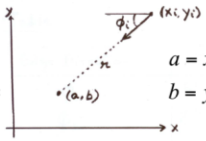
- Gradient information can save a lot of computation

if you know gradient, then gradient is red, dont need to check blue line

Edge Location (x_i, y_i)

Edge Orientation ϕ_i

If we assume radius is known: only 2 circles per gradient



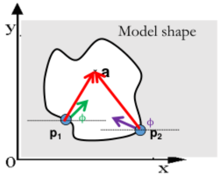
$a = x - r\cos\phi$

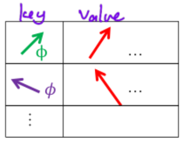
$b = y - r\sin\phi$

Generalized Hough Transform

Offline Modeling non-parametric encoding ∴ not closed form

- Define a model shape by its boundary points p_i and reference point a.
- At each boundary point p_i, compute its displacement vector from a
  - Displacement vectors vote for potential a's based on the position of p_i
- Store vectors in a table, indexed by p_i's gradient orientation ϕ



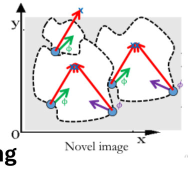


For each row in the table, compute some averaging statistics of the displacement vectors in that ϕ bin, e.g. mean / mode(s)

Detection Procedure:

For each edge point:

- Use its gradient orientation ϕ to index into stored table
- Retrieve **r** vectors (mean/mode displacement vector for that ϕ) to vote for the reference point a



Voting

Minimize irrelevant tokens first, e.g. convert to edge image

Choose a good grid / discretization

Soft voting for neighbors (smoothing effect in accumulator array)

$(\theta,\rho) \rightarrow 0.25*(\theta,\rho-1), 0.5*(\theta,\rho), 0.25*(\theta,\rho+1)$

Limit voting extent from each token

Use direction of edge to reduce amount of cast votes

Keep tags on votes to read back points contributing to local maxima

Feature Selection

Spatial

Intensity

Colour

k-Means Clustering

Basic idea: randomly initialize the k cluster centers, and iterate between assigning membership and computing cluster centers.

1. Given K, randomly initialize the cluster centers, $c_1, \ldots, c_K$
2. Given cluster centers, determine points in each cluster
   - For each point p_i, find the closest c_j. Put p_i into cluster j
3. Given points in each cluster, solve for c_j
   - Set c_j to be the mean of points in cluster j
4. If c_j have changed (up to some threshold), repeat Steps 2, 3.

Pros

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

Cons/issues

- Setting k?
- Sensitive to initial centers
- Sensitive to outliers
- Detects spherical clusters
- Assumes means can be computed (efficient, meaningful)

slope-intercept form: y=mx+b (m=>slope, b=>y-intercept)

double-intercept form: $\frac{x}{a} + \frac{y}{b} = 1$ (a=>x-intercept, b=>y-intercept)

normal form: $x\cos\theta + y\sin\theta = \rho$

y=-4x+2

y+4x=2=>y/2+x/(1/2)=1

x cos14.07+ y sin14.07 = 2/($\sqrt{17}$)

$\rho^2 = \frac{1}{\frac{1}{a^2}+\frac{1}{b^2}} = \frac{1}{\left(\frac{1}{2}\right)^2+2^2} = \frac{4}{17}$, $\rho = \frac{2}{\sqrt{17}}$

$a = \frac{\rho}{\cos\theta}$, $b = \frac{\rho}{\sin\theta}$

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

SLIC

Definitions, Features & Distance Measure

If c dominates, it will end up as hexagons as it's a tesselable circle

n_{tp}; n_{sp} — Total number of pixels in image; number of superpixels we want (total pixels → hyperparameter, normally determined with some other heuristic)

$s = [n_{tp}/n_{sp}]^{1/2}$ — Initial spacing of each superpixel (sqrt for 2d)

Features $z = [\underbrace{r,g,b}_{colour},\underbrace{x,y}_{position}]$

If you use HSV, the range of H,S,V are different so need to normalize properly

Colour Distance $d_c = \left[(r_j-r_i)^2 + (g_j-g_i)^2 + (b_j-b_i)^2\right]^{1/2}$ euc distance

Grey-scale version (intensity) $d_c = \left[(l_j-l_i)^2\right]^{1/2}$

L2 distance measures

Spatial Distance $d_s = \left[(x_j-x_i)^2 + (y_j-y_i)^2\right]^{1/2}$

Composite Distance $D = \left[\left(\frac{d_c}{d_{cm}}\right)^2 + \left(\frac{d_s}{d_{sm}}\right)^2\right]^{1/2} \Rightarrow D = \left[d_c^2 + \left(\frac{d_s}{s}\right)^2 c^2\right]^{1/2}$

see FAQ for derivation; c becomes the only hyperparameter

normalisation ensure d_c and d_s plays equally

scaling factors: set as max expected values of d_c and d_s

$d_{sm} = s = [n_{tp}/n_{sp}]^{1/2}$

Consider as hyperparameter weighing relative importance between colour similarity vs. spatial proximity.

· Since colour's [0,255] dc has a limited range

· images can be very big so d_s can be very big → so set to spacing (scales)

Initialization

centres should be at a representative pixel and we take 3×3 to be a good number

distance between centres is s

number of grids is $n_{superpixel}$

1. ***Initialize the algorithm:*** Compute the initial superpixel cluster centers,

$$\mathbf{m}_i = [r_i\ g_i\ b_i\ x_i\ y_i]^T,\ i = 1, 2, \ldots, n_{sp}$$

by sampling the image at regular grid steps, s. Move the cluster centers to the lowest gradient position in a 3×3 neighborhood. For each pixel location, p, in the image, set a label $L(p) = -1$ and a distance $d(p) = \infty$.

Different from k-means; rather than initialize randomly, initialize cluster centers on a grid.

Do not initialize cluster center on an edge or noise point (these have some more extreme values); convergence is faster if we can already initialize on a value common to the surrounding pixels, (approximate as lowest gradient position).

Update + Convergence

· 2s has no relation to the 3×3 from earlier

· 2s, because you don't need to look too far. each superpixel spans a certain range. If you look beyond too far, it's already at another super pixel

2. ***Assign samples to cluster centers:*** For each cluster center $\mathbf{m}_i$, $i = 1, 2, \ldots, n_{sp}$, compute the distance, $D_i(p)$ between $\mathbf{m}_i$ and *each* pixel p in a $2s \times 2s$ neighborhood about $\mathbf{m}_i$. Then, for each p and $i = 1, 2, \ldots, n_{sp}$, if $D_i < d(p)$, let $d(p) = D_i$ and $L(p) = i$.

Compute distance only to closest set of cluster centers.

3. ***Update the cluster centers:*** Let C_i denote the set of pixels in the image with label $L(p) = i$. Update $\mathbf{m}_i$:

Same as k-means

Post-Processing

5. ***Post-process the superpixel regions:*** Replace all the superpixels in each region, C_i, by their average value, $\mathbf{m}_i$.

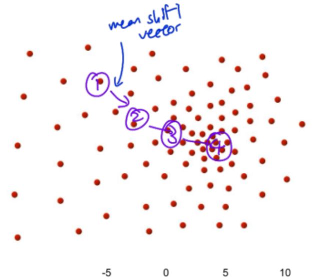
Optional, to create the "stained-glass" effect.

Mean Shift Algorithm

Main Idea: find **modes** or local density maxima in the feature space

For each data point:

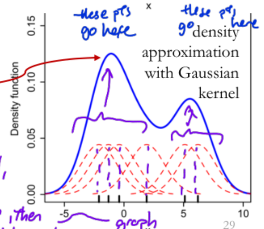
1. Define a window around it, compute the centroid
2. Shift the center of the window to the centroid
3. Repeat until the centroid stops moving (convergence).



Mean shift tries to find the local maxima in this density.

these blue and red lines have shape determined by the kernel, in this case Gaussian

if Gaussian σ is too high, then graph; if σ too low, then !!! graph



Images from https://en.wikipedia.org/wiki/Kernel_density_estimation

Mean-Shift Procedure

For each point $\mathbf{x}_j$, where j = 1…n

1. Initialize density window $\mathbf{x}$: $\mathbf{x} = \mathbf{x}_j$ (current surrounding area)
2. Compute mean shift vector $\mathbf{m}$: $\mathbf{m}_{h,G}(\mathbf{x}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i g(\ldots)}{\sum_{i=1}^{n} g(\ldots)}$

n total data indexed; size of circle; size of Gaussian

Mean shift vector is a function of the bandwidth h and kernel G. These are expressed as subscripts to keep notation tidy.

3. Shift density window and update: $\mathbf{x}' = \mathbf{x} + \mathbf{m}(\mathbf{x})$, $\mathbf{x} = \mathbf{x}'$
4. Iterate steps 2 and 3 until convergence. gradient descent

Segmentation

1. Find features (color, gradients, texture, etc) to represent pixels.
2. Initialize density windows at individual feature points.
3. Perform mean shift for each point until convergence.
4. Merge pixels whose feature points that end up near the same mode or "peak" into the same segment.

(Need to choose bandwidth/window size

Optimisations

Points along search path/end point are of same cluster

Pros

General algorithm for mode-finding (we apply it towards segmentation)

No prior assumptions on cluster shape (spherical, elliptical, etc.)

One parameter (window size / bandwidth h, which has a physical meaning)

Finds variable number of modes (vs. pre-specified k in k-means)

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