LINEAR SHIFT INVARIANT SYSTEMS

$f \longrightarrow$ LSIS $\longrightarrow g$

Linearity $f_1 \rightarrow$ LSIS $\rightarrow g_1$ $\quad f_2 \rightarrow$ LSIS $\rightarrow g_2$

$$T(\alpha f_1 + \beta f_2) = \alpha T(f_1) + \beta T(f_2) = \alpha g_1 + \beta g_2$$

Shift invariance $f(x) \rightarrow$ LSIS $\rightarrow g(x)$, shift by a, $f(x-a) \rightarrow$ LSIS $\rightarrow g(x-a)$

Convolution of two functions $f(x)$, $h(x)$

$$g(x) = f(x) * h(x) = \int_{-\infty}^{\infty} h(x-\tau) f(\tau) d\tau$$ superposition integral

Flip h, slide the function h over every point and do product integral

Separable Filter $w[r,c]$ is separable iff $w[r,c] = w[r]\,w[c]$

Any convolution is a linear shift invariant system

Linearity is trivial

Shift invariance: Let $g(x) = \int_{-\infty}^{\infty} f(\tau)\, h(x-\tau)\, d\tau$.

Then $\int_{-\infty}^{\infty} f(\tau-a)\, h(x-\tau)\, d\tau = \int_{-\infty}^{\infty} f(\mu)\, h(x-a-\mu)\, d\mu = g(x-a)$

substitute $\mu = \tau - a$, $d\mu = d\tau$

Given a LSIS black box, can we find h? $\quad f \longrightarrow \boxed{h} \longrightarrow g(x) = \int_{-\infty}^{\infty} f(\tau)\, h(x-\tau)\, d\tau$

What input f will produce an output $g = h$?

Unit Impulse Function is infinitesimally thin & tall. As $\varepsilon \to 0$, it gets thinner & taller.

$$\delta(x) = \begin{cases} 1/2\varepsilon & |x| \le \varepsilon \\ 0 & |x| > \varepsilon \end{cases} \qquad \int_{-\infty}^{\infty} \delta(\tau)\, d\tau = \frac{1}{2\varepsilon} \cdot 2\varepsilon = 1$$ area is always 1

Let $g(x) = \int_{-\infty}^{\infty} \delta(\tau)\, h(x-\tau)\, d\tau = h(x)$.

Notice that "flipping" a unit impulse function gives us the same thing.

when sliding h over, we are integrating with an infinitesimally small section w/ area of 1, giving us $h(x)$

This is called the sifting property

Therefore, all we need to do is hit it with a unit impulse function $\quad \delta \rightarrow \boxed{h} \rightarrow h$ (unit impulse → impulse response)

$$h(x) = \delta(x) * h(x) \qquad h(x) = \int_{-\infty}^{\infty} \delta(\tau) h(x-\tau)\, d\tau$$

Properties of Convolution

Commutative $\quad a * b = b * a$

Associative $\quad (a * b) * c = a * (b * c)$

Cascaded system $\quad f \rightarrow h_1 \rightarrow h_2 \rightarrow g = f \rightarrow h_1 * h_2 \rightarrow g = f \rightarrow h_2 * h_1 \rightarrow g$

2D convolution

Continuous $\quad g(x,y) = \iint_{-\infty}^{\infty} f(\tau,\mu)\, h(x-\tau, y-\mu)\, d\tau\, d\mu$ also extends to 3D

Discrete $\quad g(x,y) = \sum_{m=1}^{M} \sum_{n=1}^{N} f(m,n)\, h(x-m, y-n)$

L4

NON-LINEAR IMAGE FILTERS

Gaussian

Gaussian kernel is a smoothing filter defined by

$$n(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$ where σ is the standard deviation

The larger σ, the more smoothing / blurring you see.

Gaussian smoothing is separable (separates, exponent rules)

$$g(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty}\int e^{-\frac{\tau^2+\mu^2}{2\sigma^2}} f(x-\tau, y-\mu)\, d\tau\, d\mu$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} e^{-\frac{\mu^2}{2\sigma^2}} \int_{-\infty}^{\infty} e^{-\frac{\tau^2}{2\sigma^2}} f(x-\tau, y-\mu)\, d\tau\, d\mu$$

Using one 2D Gaussian filter is equivalent to using two 1D Gaussian filters

mask is K^2, 2D has K^2 multiplications & K^2-1 additions, 1D has $2K$ multiplications & $2(K-1)$ additions for one pixel

Problem w/ smoothing does not remove outliers (noise)
smooths edges (blur

If the same Gaussian kernel is used everywhere, the edges become blurry

Bilateral Filtering

"Bias" Gaussian kernel s.t. pixels not similar in intensity to center pixel receive lower weight.

$$g(x,y) = \int_{-\infty}^{\infty}\int G_{\sigma_s}(x-\tau, y-\mu)\, G_{\sigma_b}(f(\tau,\mu)-f(x,y))\, f(\tau,\mu)\, d\tau\, d\mu$$

G_{σ_s}: Spatial Gaussian — distance from center
G_{σ_b}: Brightness Gaussian — difference in intensity — gives higher weight to pixels similar in intensity

- $\sigma_s \uparrow$ shaded regions, regions similar in colour, get flatter & flatter
- $\sigma_b \uparrow$ brightness Gaussian becomes flat, has no effect, bilateral behaves like G_{σ_s} filter

FOURIER TRANSFORM

Any periodic sum can be written as a weighted sum of infinite sinusoids of different frequencies

x = space, u = frequency, $e^{i\theta} = \cos\theta + i\sin\theta$, $i = \sqrt{-1}$

The Fourier Transform (FT) represents a signal $f(x)$ in terms of amplitudes & phases of its constituent sinusoids

$$f(x) \longrightarrow \boxed{FT} \longrightarrow F(u) \qquad F(u) = \int_{-\infty}^{\infty} f(x)\, e^{-i2\pi ux}\, dx$$

The Inverse Fourier Transform (IFT) computes signal $f(x)$ from amplitudes & phases of its constituent sinusoids.

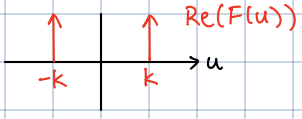
$$f(x) \longleftarrow \boxed{IFT} \longleftarrow F(u) \qquad f(x) = \int_{-\infty}^{\infty} F(u)\, e^{i2\pi ux}\, du$$

$f(x)$ = sum of sinusoids of different frequencies (u)
& amplitudes captured by Fourier coefficient $F(u)$

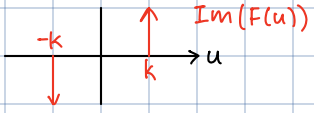
$F(u)$ holds the amplitude & phase of sinusoid of frequency u.

$$F(u) = \text{Re}(F(u)) + i\,\text{Im}(F(u))$$

↳ $f(x) = \cos(2\pi kx)$, $\cos x$ w/ period $\frac{1}{k}$
$\Rightarrow F(u) = \frac{1}{2}[\delta(u+k) + \delta(u-k)]$



↳ $f(x) = \sin(2\pi kx)$
$\Rightarrow F(u) = \frac{1}{2} i\,[\delta(u+k) - \delta(u-k)]$



↳ $f(x) = \delta(x) \Rightarrow F(u) = 1$, $f(x) = 1 \Rightarrow F(u) = \delta(u)$

↳ $f(x) = \text{Rect}\left(\frac{x}{\varepsilon}\right)$, pulse w/ width ε centered at 0
$\Rightarrow F(u) = \varepsilon\,\text{sinc}(\varepsilon u) = \varepsilon\,\frac{\sin(\varepsilon u)}{\varepsilon u}$ oscillations that dampen as $u \to \pm\infty$

↳ $f(x) = e^{-x^2/2\sigma^2}$ Gaussian $\Rightarrow F(u) = \sqrt{2\pi\sigma^2}\, e^{-2\pi^2\sigma^2 u^2}$? Another Gaussian, but w/ smaller standard deviation
Widening a Gaussian in spatial domain ($\sigma\uparrow$) compresses it in the frequency domain

Properties

	Spatial Domain	Frequency Domain
Linearity	$\alpha f_1(x) + \beta f_2(x)$	$\alpha F_1(u) + \beta F_2(u)$
Scaling	$f(ax)$	$\frac{1}{\lvert a\rvert} F\left(\frac{u}{a}\right)$
Shifting	$f(x-a)$	$e^{-i2\pi ua} F(u)$
Differentiation	$\frac{d^n}{dx^n} f(x)$	$(i2\pi u)^n F(u)$

LS 5 & 6 I think idk

Convolutions and Fourier Transform

Recall, a convolution of two functions f & h is given by $g(x) = f(x) * h(x) = \int_{-\infty}^{\infty} h(x-\tau) f(\tau)\, d\tau$

The Fourier Transform of $g(x)$ is

$$G(u) = \int_{-\infty}^{\infty} g(x) e^{-i2\pi ux}\, dx$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x-\tau) f(\tau) e^{-i2\pi ux}\, d\tau\, dx \qquad e^{-i2\pi ux + i2\pi ux\tau - i2\pi ux\tau}$$

$$= \int_{-\infty}^{\infty} h(x-\tau) e^{-i2\pi u(x-\tau)}\, dx \int_{-\infty}^{\infty} f(\tau) e^{-i2\pi u\tau}\, d\tau$$

$H(u)$ after substitution $\qquad F(u)$

Convolution of $f*h$ in spatial domain corresponds w/ multiplication of $FT(f)\, FT(H)$ in the the frequency domain

Spatial		Frequency
$g(x) = f(x) * h(x)$	$\Longleftrightarrow$	$G(u) = F(u)H(u)$
convolution		multiplication

$g(x) = f(x)\, h(x)$	$\Longleftrightarrow$	$G(u) = F(u) * H(u)$
multiplication		convolution

$$\begin{array}{ccccc} g(x) & = & f(x) & * & h(x) \\ \uparrow \text{IFT} & & \downarrow \text{FT} & & \downarrow \text{FT} \\ G(u) & = & F(u) & \times & H(u) \end{array}$$

Watch Convolution Theorem vid for visual example

Transform image into frequency domain, pass a filter, IFT to see result of that filter in spatial domain

- Low-Pass Filter keep low frequencies (constant intensities), eliminate high frequencies (sharp edges)
  - keep frequencies in the center
  - $\Rightarrow$ blurrier image, smaller disc in frequency domain makes it even blurrier
- High-Pass Filter keep high frequencies (cutout the low frequencies in the center)
  - $\Rightarrow$ remove constant brightness regions, left w/ sharper edges
  - larger cutout $\Rightarrow$ more pronounced edges
- Band-pass Filter keep high & low frequencies
  - $\Rightarrow$ keep constant intensity areas and edges

Gaussian Smoothing Convolving image w/ Gaussian $\equiv$ multiply Gaussian w/ frequency of image, IFT back to spatial

σ_s spatial $\propto \frac{1}{\sigma_f}$ frequency (inversely proportional)

L5?

SAMPLING THEORY AND ALAISING

Lens of camera creates a continuous optical image on the image plane. The sensor converts continuous → discrete

How densely should we sample the continuous image to not lose important information & remove artifacts?

Aliasing undersampling the signal, either losing frequencies or introducing unwanted patterns
↳ Movie patterns

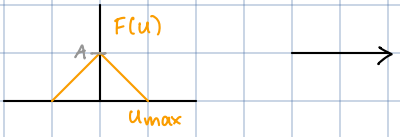
we sample a continuous signal by multiplying w/ the **Shah Function (Impulse Train)** train of δ functions

Spatial Domain $s(x) = \sum_{n=-\infty}^{\infty} \delta(x - n\Delta x)$ $\quad f_s(x) = f(x)s(x)$
↑ displacement until the next sample

Fourier Domain $S(u) = \frac{1}{\Delta x} \sum_{n=-\infty}^{\infty} \delta\left(x - \frac{n}{\Delta x}\right)$ displacement = $\frac{1}{\Delta x}$

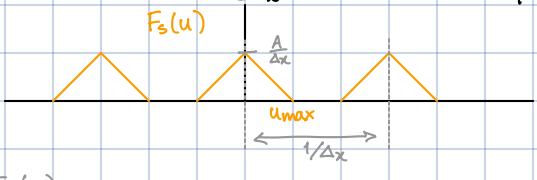
Convolving any function w/ δ function gives us the same function
$F(u) * S(u)$ gives us multiple copies of $F(u)$ where the spacing between copies is $\frac{1}{\Delta x}$

$u_{max} = \max u \text{ s.t. } F(u) > 0$



→

we need $u_{max} \leq \frac{1}{2\Delta x}$, otherwise the frequencies overlap & we get aliasing



$F(u)$ from $F_s(u)$

Nyquist Theorem We can recover $f(x)$ from $f_s(x)$ only if $u_{max} \leq \frac{1}{2\Delta x}$. This is called the **Nyquist Frequency**.
Then the continuous signal is well sampled. If $u_{max} > \frac{1}{2\Delta x}$, the image is undersampled.

Let's say the signal is well-sampled. To get $f(x)$

Cutout one copies $C(u) = \begin{cases} \Delta x & \text{if } |u| < \frac{1}{2\Delta x} \\ 0 & \text{otherwise} \end{cases}$,

multiply by FT to get $F(u) = F_s(u)C(u)$ gives us one signal

$f(x) = IFT(F(u))$

Aliasing occurs ⊙ imaging a scene (signal) that has frequencies above image sensor's Nyquist Frequency
Some image sensors use a low-pass filter, which cuts out the high frequencies.
Another solution is to average samples in an area (supersampling?) to smooth the image.

Colour

Our eyes have 3 receptors (cone cells) that respond to visible light, tri-stimulus response

Objects reflect different wavelengths of light, described by a spectral power distribution (SPD)
shows relative amount of each λ is reflected over visible spectrum.

Grassman's law a source colour can be matched by a linear combination of 3 independent "primaries"
Light obeys laws of linear algebra

Each SPD goes through a "black box" in the human brain.
CIE RGB human subjects matched colours they saw with λ, creating a "standard"
Two different SPDs can be preceived as the same colour, SPDs are called metamers

$$r_1 = \int f_1(\lambda)\, r(\lambda)\, d\lambda \quad = \int f_2(\lambda)\, r(\lambda)\, d\lambda \quad = r_2$$
$$g_1 = \int f_1(\lambda)\, g(\lambda)\, d\lambda \quad = \int f_2(\lambda)\, g(\lambda)\, d\lambda \quad = g_2$$
$$b_1 = \int f_1(\lambda)\, b(\lambda)\, d\lambda \quad = \int f_2(\lambda)\, g(\lambda)\, d\lambda \quad = b_2$$

If the stimulus is the same, then we preceive as same colour

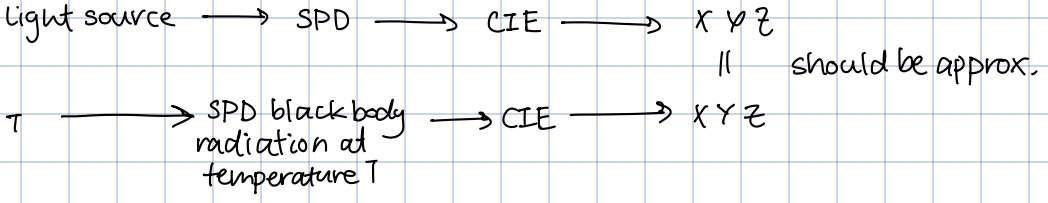
Bayer pattern used in most camera sensors —— 317
Every camera has a different RGB filter, sensitive to different sensors.
Colour filter array (CFA) on camera filters light into 3 sensor-specific RGB primaries

Raw RGB sensor images are <u>not</u> in standard colour space
There is device specific mapping to get to standard response (CIE XYZ)

Colour Constancy

In a real scene, an object's SPD is a combination of its reflectance properties & scene illumination
Our visual system is smart, but image sensors cannot compensate

Different illuminants have different SPDs. Lighting industry uses colour temperature (↓ warm, ↑ cool)



Light sources can have same colour temp, but different SPD. If the SPD map to the same CIE XYZ as the temperature black box, then they will be preceived as same colour

A white point is a CIE XYZ value of ideal "white target", an illuminants SPD in terms of CIE XYZ
In colour correcting, we make white points the same between scenes

Illuminant to illuminant mapping used to compensate for standard illumination condition of which image will be viewed

⊖ mapping CIE XYZ to colour space, specify the white point

Displaying Colour

Each display device has its own SPDs. We need to know its colour space to display colours accurately.
Display drivers perform conversion to output colour space

$$M_1 \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} L_1 \\ M_1 \\ S_1 \end{bmatrix} \qquad M_2 \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} L_2 \\ M_2 \\ S_2 \end{bmatrix}$$

so now even on different screens, the colour looks the same

We can also do illuminant to illuminant mapping to compensate for ambient lighting in displays?
Some computers auto adjust based on light setting

Human preception of brightness is not linear $S = k I^a$

Use gamma-correction to adjust brightness accordingly $I \propto V_s^{\gamma}$ gamma-encoded intensity

DIFFERENTIABLE 1D & 2D REPRESENTATIONS

Derivative filters can be used for edge detection

Edge enhancement / detection — mathematical operations on intensity

Boundary detection — account for human visual perception of what is / isn't an object

Fitting Models to Data

Recall, function interpolation constructs a new function $g(x)$ from discrete samples of $f(x)$ at $x_1, \dots, x_n$ that satisfies $g(x) = f(x)$, can be evaluated $\forall x \in \mathbb{R}$.

Passes through all samples of f

Polynomial fitting just gets it close enough, approximates it.

$p(x)$ minimizes distance from all samples within a window

WLOG, assume window's central sample $x_{w+1} = 0$.

Taylor Expansion

$$f(x) = f(0) + x\frac{df}{dx}(0) + \frac{1}{2}x^2\frac{d^2f}{dx^2}(0) + \cdots + \frac{1}{n!}x^n\frac{d^nf}{dx^n}(0) + R_{n+1}(x)$$

residual

$\lim_{x \to 0} R_{n+1}(x) = 0$

n^{th} order approx of f

For n-degree polynomial with $2w+1$ samples:

- $2w+1$ equations
- n unknowns

$$\underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \end{bmatrix}}_{\text{known}} = \underbrace{\begin{bmatrix} 1 & x_1 & \frac{1}{2}x_1^2 & \frac{1}{6}x_1^2 & \cdots & \frac{1}{n!}x_1^n \\ 1 & x_2 & \frac{1}{2}x_2^2 & \frac{1}{6}x_2^2 & \cdots & \frac{1}{n!}x_2^2 \\ \vdots & & & & & \end{bmatrix}}_{\text{known}} \underbrace{\begin{bmatrix} f(0) \\ \frac{df}{dx}(0) \\ \vdots \\ \frac{d^nf}{dx^n}(0) \end{bmatrix}}_{\text{unknown}}$$

$$b_{k\times 1} = A_{k\times n}\, d_{n\times 1}$$

If $k = n$, we can solve system perfectly.

$k > n$ (more rows than columns), overdetermined linear system. Might not always be able to solve, but we can minimize error

Solve system in terms of d that minimizes total squared error $\|b - Ad\|^2$

According to the Taylor approximation, the solution of linear system provides samples of the n derivatives of $f(x)$ at the window's central sample

$$d \approx \begin{bmatrix} f(0) \\ \vdots \\ \frac{d^nf}{dx^n}(0) \end{bmatrix}$$

Basic Moving Least Squares Algorithm

1. Set $k = 1$ (sample)
2. Define a "sample window" $x_k, \dots, x_{2w+k}$ centered at sample x_{w+k}
3. Fit an n-degree polynomial $p(x)$ to the datapoints $(x_k, f_k), \dots, (x_{2w+k}, f_{2w+k})$
4. Use $p(x)$ to evaluate $f(x)$ and its derivatives at $x = x_{w+k}$
5. Set $k = k+1$ and repeat steps 2-4 until last sample is reached.

Parametric 2D Curve Representations

A continuous mapping $\gamma: [a,b] \to \mathbb{R}^2$ maps a curve parameter $t \in [a,b]$, the position along the curve, to a point on the curve $(x(t), y(t))$, such that $\gamma(a) = \gamma(b)$

↳ boundary curve, t = pixel, $x(t)$ = x coord of t^{th} pixel, $y(t)$ = y coord of t^{th} pixel

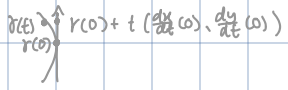
To fully describe a curve, we need the coordinate functions $x(t), y(t)$

The curve is smooth ⊙ all derivatives of coordinate functions, $\frac{d^n x}{dt^n}(t)$, $\frac{d^n y}{dt^n}(t)$, exist for all n, t

The first & 2nd derivatives are informative about the curve's shape.

The mapping $\gamma(t)$ is a vector-valued function if $\gamma(t)$ maps t to a 2D point $(x(t), y(t))$

The tangent vector at $\gamma(t)$ is $$\frac{d\gamma}{dt}(t) = \left(\frac{dx}{dt}(t), \frac{dy}{dt}(t)\right)$$



The 1st-order Taylor series approximation of $\gamma(t)$ near $\gamma(0)$:

$$\gamma(t) \cong \left(x(0) + t\frac{dx}{dt}(0), y(0) + t\frac{dy}{dt}(0)\right) = \underbrace{(x(0), y(0))}_{\text{point}} + t\underbrace{\left(\frac{dx}{dt}(0), \frac{dy}{dt}(0)\right)}_{\text{tangent vector}}$$

γ can be parametrized in many different ways. Regardless of the parametrization, the direction is the same.

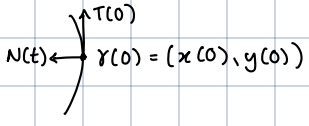
The unit tangent is the tangent but length = 1

$$T(t) = \frac{d\gamma}{dt}(t) \cdot \frac{1}{\left\|\frac{d\gamma}{dt}(t)\right\|}$$

Does not depend on choice of parameter t.

The unit normal is given by

$$N(t) = \frac{1}{\left\|\left(\frac{dy}{dt}, \frac{dx}{dt}\right)\right\|_2}\left(-\frac{dy}{dt}(t), \frac{dx}{dt}(t)\right)$$



orthogonal to tangent $T(t)$ $\quad N(t) \cdot N(t) = 0$

corresponds to a 90° counter-clockwise rotation

The moving frame is the pair of orthogonal vectors $\{T(t), N(t)\}$, the tangent and normal.

As we change parameter t, the faster the moving frame rotates the more "curved" the curve is.

Using first-order Taylor series near $t=0$

$$\{T(t), N(t)\} = \{T(0), N(0)\} + \left\{t\frac{dT}{dt}(0), t\frac{dN}{dt}(0)\right\}$$

↳ circle $\gamma(\theta) = r(\cos\theta, \sin\theta)$

$$\frac{d\gamma}{d\theta}(\theta) = r(-\sin\theta, \cos\theta), \quad T(\theta) = (-\sin\theta, \cos\theta), \quad N(\theta) = (-\cos\theta, -\sin\theta)$$

Local Differential Analysis for Edge Detection

Locally, the intensity $f(x,y)$ near $(0,0)$ can be expressed as a polynomial of x & y

$$f(x,y) = f(0,0) + x\frac{\partial f}{\partial x}(0,0) + y\frac{\partial f}{\partial y}(0,0) + R(x,y)$$

Consider a 1D slice of the image trough $(0,0)$ in the direction of unit vector $v = (\cos\theta, \sin\theta)$

Express slice as $g(t) = f(t\cos\theta, t\sin\theta)$ $\forall t \in \mathbb{R}$, for fixed θ

The 1st-order approx. from Taylor series is $g(t) = f(t\cos\theta, t\sin\theta) = f(0,0) + t\underbrace{\left(\cos\theta\frac{\partial f}{\partial x}(0,0) + \sin\theta\frac{\partial f}{\partial y}(0,0)\right)}_{\text{derivative of } g(t)}$

The directional derivative of f at (a,b) in direction $v \in \mathbb{R}$ is given by

$$D_v f = \left[\frac{\partial f}{\partial x}(a,b) \quad \frac{\partial f}{\partial y}(a,b)\right] v = \nabla f(a,b) \cdot v$$

maximized @ ∇f is along v, minimized @ ∇f is orthogonal to v ($v \perp \nabla f$)

$D_v f(x,y)$ tells us how far "up" or "down" we will go if we move away from (a,b) by an infitesimal amount in the direction v.

The image gradient is $\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\right)$.

∇f is in direction of steepest change,

orthogonal to direction where image intensity remains constant

$\nabla f(x,y)$ is the normal vector of the isophote curve through (x,y) (equal brightness)

orientation: $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y}(x,y) \Big/ \frac{\partial f}{\partial x}(x,y)\right)$ magnitude: $|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}(x,y)\right)^2 + \left(\frac{\partial f}{\partial y}(x,y)\right)^2}$

A simple edge detector uses a threshold on the gradient magnitude.

Define two thresholds: g_{min}, g_{low}

Mark as an edge pixel if the pixel has $|\nabla I| > g_{min}$.

Also mark as an edge for pixels w/ $|\nabla I| > g_{low}$ if they are adjacent to already-marked edge pixel

Strong gradients are more likely to be an edge

Weaker gradients more likely to be on an edge if they are near a strong gradient

Including the weaker gradients reduces the chance of "broken" edges.

L9 Provides edge normal info, less sensitive to image noise

Recall the image gradient $\nabla f(x,y)$ tells us how quickly intensity changes in neighbourhood of (x,y)

We can model an edge as a smooth step function, inflection point defines edge's location

Inflection point local extremum of 1st derivative, zero-crossing of 2nd derivative

The image Laplacian is

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

scalar function of (x,y), analog to 2nd derivative

Its zero-crossings can be used to localize edges

Laplacian of the Gaussian filter

$$\nabla^2 G_\sigma(x,y) = \frac{\partial^2 G_\sigma}{\partial x^2}(x,y) + \frac{\partial^2 G_\sigma}{\partial y^2}(x,y)$$ scaled difference of Gaussians

Basic approach for edge detection using $\nabla^2 f$:

1. Compute $f * \nabla^2 G_\sigma$
2. Detect zero-crossings of $f * \nabla^2 G_\sigma$

   pixel (x,y) for which $f * \nabla^2 G_\sigma$ has different sign from at least one pixel in a 3×3 neighbourhood
3. Prune zero-crossings that are not significant

   difference between a positive & negative in 3×3 neighbourhood is below a threshold, removes noise

Threshold only for rejection of weak edges, excellent localization, sensitive to noise

Non-maxima suppression

1. Compute $\nabla I(q)$ at each pixel q
2. Mark each q as an edge only if $|\nabla I|$ is highest at q compared to neighbouring pixels, p, r, in the same direction as $\nabla I(q)$

   Compute $|\nabla I(p)|$ & $|\nabla I(r)|$ by interpolation if p & r are not on exact pixels.

Canny's Edge Detection Algorithm

1. Compute $\frac{\partial G_\sigma}{\partial x} * f$, $\frac{\partial G_\sigma}{\partial y} * f$ for chosen scale parameter σ
2. Compute gradient magnitude & gradient orientation at each pixel.
3. Perform non-maximum supression

   Only identify pixels that have max ∇ magnitude, makes edges thin
4. Perform hysteresis thresholding

   all pixels that have gradient magnitude $> g_1^{min}$, pixels that neighbour edge pixel w/ gradient $> g_2^{low}$

Applications

Intelligent scissors lasso finds boundary of an object following the user's mouse

1. User specifies a seed point on the object boundary
2. Algorithm automatically computes a live wire (boundary) that connects seed to current mouse position
3. Segments can be locked in place by defining a new seed at the mouse current position

Imagine image as a graph, pixels are vertices. Between each neighbouring pixel is a link/edge w/ weight.

Goal: find shortest-weight path connecting seed & mouse

Let link weight $l(p,q)$ be defined s.t. links along an image edge have very low weight.

$$l(p,q) = 0.43\, f_z(q) + 0.43\, f_D(p,q) + 0.14 f_G(q)$$

① ② ③

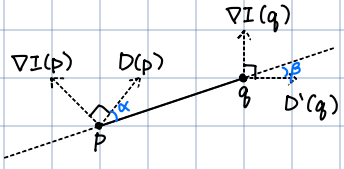
① Laplacian zero-crossing term

$$f_z(q) = \begin{cases} 0 & \text{if } q \text{ is on a zero-crossing} \\ 1 & \text{otherwise} \end{cases}$$

(on a potential edge, lower weight)

② Edge direction consistency term

If p & q are on the same edge, their gradients will point in similar directions



$$f_D(p,q) = \frac{\alpha + \beta}{\pi}$$

smaller α & β if on the same edge

③ Gradient term

$$f_G(q) = \frac{\max(G) - G(q)}{\max(G)}$$

$G(q)$: gradient magnitude at q

$\max(G)$: max gradient magnitude over image

Use Dijstra's algorithm to pre-compute min-cost path from seed to all image pixels

Painterly Rendering giving images "painted" finish

Draw strokes at same angle, length is random

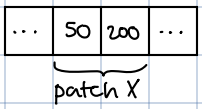
Edge detection to prevent strokes from crossing edges

Re-orient strokes by changing angles to be perpendicular to gradient (parallel to edge), interpolate if needed.

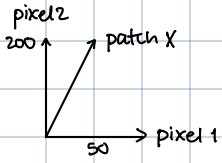
VECTOR IMAGE REPRESNTATIONS

We can represent discrete pixel neighbours as vectors

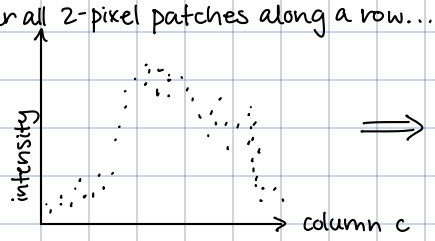
Each patch X_i is the intensities of 2 neighbouring pixels

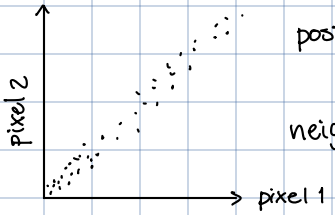


$X = \begin{bmatrix} 50 \\ 200 \end{bmatrix}$



If we gather all 2-pixel patches along a row...





positive correlation between adjacent pixels

neighbouring pixels have similar intensities

also applies for 3D

In general, $k \times l$ neighbourhoods represented as kl-dimensional vectors

Template Matching

Goal given a template T, find its occurrences (if any) in an image

$\Rightarrow$ find image patch X_{rc} that is most similar to given template T, $\underset{(r,c)}{\text{argmax}}\ sim(T, X_{rc})$ (similarity of T & X_{rc})

(image patch: vector)

1. Define array similarity( ) of size equal to image.
2. Set similarity$(r,c) = sim(T, X_{rc})$ for every pixel (r,c) that is computable.
3. Search array similarity( ) for pixel (r,c) w/ highest similarity score

Matching success highly dependent on similarity function

(Squared) Euclidean Distance $sim(T, X_i) = -\|X_i - T\|^2$ (negative b/c we use argmax)

Can be computed as a 2D sum for a patch centered at pixel (r,c) w/ template width $2N+1$

$$sim(r,c) = -\sum_{a=-N}^{N}\sum_{b=-N}^{N}(I(r+a,c+b) - T(a,b))^2$$

Cannot distinguish between patches that are just scaled versions of T (brighter/dimmer) from other patches

Cross-correlation (CC) dot product, $CC(T, X_i) = X_i^T \cdot T = \|X_i^T\| \cdot \|T\| \cos\theta$ (matrix)

Depends on lengths of $X_i \cdot T$. Among all X_i of same length, maximize θ X_i & T have same direction.

$$CC(r,c) = \sum_{a=-N}^{N}\sum_{b=-N}^{N} I(r+a,c+b)\,T(a,b)$$

Normalized Cross-correlation (NCC) cosine of angle between vectors $NCC(T, X_i) = \dfrac{X_i^T \cdot T}{\|X_i\| \cdot \|T\|} = \cos\theta$

Independent of lengths of X_i & T. Maximized @ X_i, T identical up to a scalar factor.

NCC = CC @ $\|X_i\| = \|T\| = 1$

Given a template w/ M pixels, we have a M-dimensional column vector. Let the image we are searching have N pixels.

CC has M multiplications, M-1 additions per patch, for a total of $O(M \cdot N)$ operations for entire image.

Principal Component Analysis

Dimensionality reduction represent T, X_i w/ $d << M$ coordinates to improve efficiency to $O(dN)$

Let X_i be a two-pixel patch from a single image row.

Recall that the coordinates are related $\Rightarrow$ can be represented more efficiently.

Dimensionality reduction allows $X_1, \ldots, X_n$ to be represented w/ fewer dimensions w/o significant information loss.

1. Find optimal set of basis vectors $B_1, \ldots, B_M$ (principal components, eigenfaces)
2. Compute patch coordinates in that basis
3. Discard axes w/ near zero coordinates for all patches (keep eigenvectors w/ largest eigenvalues)

Let X be the patches. Let B be the basis matrix. Let Y be the representation of X in basis B.

$$[X_1 \; X_2 \; \cdots \; X_N] = [B_1 \; B_2 \; \cdots \; B_M] \begin{bmatrix} y_1^1 & y_2^1 & \cdots & y_N^1 \\ y_1^2 & y_2^2 & \cdots & y_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ y_1^M & y_2^M & \cdots & y_N^M \end{bmatrix} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \approx 0, \text{ can discard}$$

Goal choose $B_1, \ldots, B_M$ s.t. $y_i^j \cong 0 \; \forall \, d < j \leq M$

so we only keep $B_1, \ldots, B_d$, $d \leq m$

Properties of Eigenvectors & Eigenvalues $Hv = \lambda v$, λ = eigenvalues, v = eigenvectors

If H is symmetric, H will have orthogonal eigenvectors

The sum of eigenvalues of a matrix is equal to its trace (2D matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\lambda_1 + \lambda_2 = a + d$)

Product of eigenvectors = determinant ($\lambda_1 \lambda_2 = ad - bc$)

PCA Algorithm Input: d (desired dimension), matrix X; Output: unit basis vectors $B_1, \ldots, B_d$

1. Compute average patch $\bar{X} = \frac{1}{N} \sum_i X_i$
2. Subtract average from each X_i, $Z_i = X_i - \bar{X} \longrightarrow Z = [Z_1 \; Z_2 \; \cdots \; Z_N]$
3. $B_1, \ldots, B_d$ are the eigenvectors of matrix ZZ^T w/ the d largest eigenvectors.

The j^{th} coordinate is significant if it varies a lot, high variance $\sigma_j^2 = \frac{1}{N} \sum_i^N (x_i^j - \bar{x}^j)^2$

@ the variance is low, coordinates are similar to each other $\Rightarrow$ redudant, can be represented w/ a single value

Covariance of vector coordinates $\sigma_{jk} = \frac{1}{N} \sum_i^N z_i^j \cdot z_i^k = \frac{1}{N} \sum_i^N r^j (r^k)^T$

- $\sigma_{jk} > 0$ the j^{th} & k^{th} coordinates are $\oplus$-coordinated
- < 0 negatively correlated
- $= 0$ no pattern/correlation

Goal of PCA is to find optimal basis $B_1, \ldots, B_M$ s.t. any two distinct rows of Y have 0 covariance (unrelated coords)

$\Rightarrow YY^T$, covariance matrix, is diagonal

Deployment (Image Recognition)

1. Compute coordinates of X in new basis $Y_i^j = B_j^T (X_i - \bar{X}) = B_j^T Z_i$
2. Compute T in basis $t^j = B_j^T (T - \bar{X})$
3. Find the vector Y_i most similar to t.
4. Return image X_i

MULTISCALE IMAGE REPRESENTATIONS

Multiscale (multiresolution) representations explicitly represent an image at multiple resolutions simultaneously.

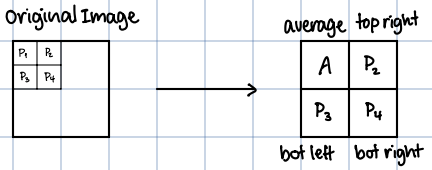
Number of distinct resolutions varies depending on representation

↳ Networks send low res versions of images first, and incrementally increase the res

lossy compression, pyramid image blending, pyramid-based video processing, scale-invariant computer vision

Minimal Invertible Multiscal Representations

no "wasted" pixels, multiple scales represented simultaneously, invertible, linear



and then recurse on the average A

until A is just 1 pixel

To invert, $A = \frac{1}{4}(P_1 + P_2 + P_3 + P_4) \longrightarrow P_1 = 4A - P_2 - P_3 - P_4$

The (discrete) wavelet transform maps an image into another basis defined by a "special" matrix B

compact representation, invertible & orthogonal, captures scale, image independent

Unnormalized Haar Wavelet Transform

↳ N = 3 Input image I^3 [9 7 3 5 6 10 2 6] The j^{th} level pyramid contains 2^j pixels.

Average I^2 [8 4 8 4] [1 -1 -2 -2] Detail coeffs. D^2

average of groups of 2 — average of group of 2 minus second value or $\frac{1}{2}$(first − second)

Average I^1 [6 6] [2 2] Detail coeff. D^1

Average I^0 [6] [0]

$$I_i^j = \frac{1}{2}\left(I_{2i}^{j+1} + I_{2i+1}^{j+1}\right)$$

filter $[\frac{1}{2}\ \frac{1}{2}]$

$$D_i^j = I_{2i}^{j+1} - \frac{1}{2}\left(I_{2i}^{j+1} + I_{2i+1}^{j+1}\right) = \frac{1}{2}\left(I_{2i}^{j+1} - I_{2i+1}^{j+1}\right)$$

filter $[\frac{1}{2}\ -\frac{1}{2}]$

The wavelet-transformed image is I^0 D^0 D^1 D^2

[6 0 2 2 1 -1 -2 -2]

matrix W (positive, negative)

$$\begin{bmatrix} I^0 \\ D^0 \\ D^1 \\ D^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{8} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \frac{1}{8} & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \frac{1}{4} & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ \frac{1}{2} & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \text{input} \\ \text{image} \end{bmatrix}$$

wavelet transformed image

Each scale j is represented by 2^j rows $\psi_0^j, \ldots, \psi_{2^j-1}^j$. Each row ψ_i^j has $\frac{2^N}{2^j} = 2^{N-j}$ non-zero elements

By multiplying I w/ W, we obtain a representation of image in basis vectors $\psi_0^0, \psi_0^1, \dots, \psi_j^i$ that form an orthogonal basis of $\mathbb{R}^2$

Notice that

$$I = \begin{bmatrix} & W & \end{bmatrix} \begin{bmatrix} & W^T & \end{bmatrix} \begin{bmatrix} \ddots & & 0 \\ & 2^{N-j} & \\ 0 & & \ddots \end{bmatrix} \lambda^{-1}$$

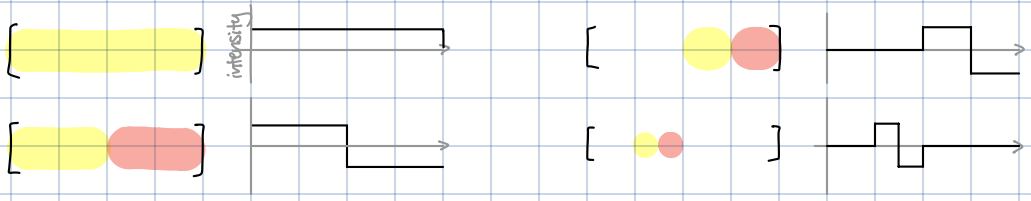
diagonal matrix

$$\Rightarrow W^{-1} = W^T \lambda^{-1}$$

W^{-1} has exactly same structure as W^T, except columns are scaled by different powers of 2

The original image of I is expressed as a weighted combination of columns of W^{-1} (or rows of W)

Each such column is called a Wavelet



non-periodic, non-zero in a bounded interval, not smooth (Fourier is opposite of everything)

$\Rightarrow$ bound signals $\Rightarrow$ bounded Haar coefficients

We can normalize W by multiplyin w/ $\lambda^{-\frac{1}{2}}$ from the left to turn it into an orthonormal matrix.

This matrix $\tilde{W} = \lambda^{-\frac{1}{2}} W$ is called the Haar wavelet transform matrix.

To normalize when calcuting I_i^j and D_i^j, divide by $\sqrt{2}$ instead of 2

$$\begin{cases} I_i^j = \frac{1}{\sqrt{2}} \left(I_{2i}^{j+1} + I_{2i+1}^{j+1} \right) \\ D_i^j = \frac{1}{\sqrt{2}} \left(I_{2i}^{j+1} - I_{2i+1}^{j+1} \right) \end{cases}$$

Application: Wavelet Image Compression

Input: 1D image I, desired compression rate k, Output: $k \cdot 2^N$ ~ coeffients (higher compression rate $\Rightarrow$ better approximation)

1. Compute $\tilde{W} I$
2. Sort wavelet coefficients in decreasing order of absolute value.
3. Keep top $k \cdot 2^N$ coefficients

Algorithm gives best least-squares approx. of the image for the given compression

Fix approx error by in step 3, only keep coefficients s.t. $|\hat{I} - I| < \varepsilon$ for error threshold ε.

For the 2D Haar wavelet transform,

1. Compute 1D linear transform of each row & store result in a new image I'
2. Compute 1D Haar transform of each column

Some other applications of wavelet include:

edge enhancement using the inverse transform

↳ denoising (maybe w/ thresholds)

Feature-based Image Matching

Goal identify "features" or patches in image I that appear in another image I'

Template matching is inefficient and needs to account for various transformations

General structure: detect features in I & I', match features across the two images

Possible errors: false positives, false negatives ——— want to minimize both

Can look at porportion of true positives vs. false positives using a Receiver Operating Characteristic (ROC) curve

To be most useful, the feature detector & matching algorithm must be insensitive to wide range of image transformations

Scale Invariant Feature Transform

SIFT is a technique for image matching that can match images invariant to scaling, rotation, affine transformations

detect keypoints from I & I', build descriptors for each keypoint, match features based on descriptors

What is a keypoint? Should have rich image content, well-defined representation, well-defined position, invariant to rotation & scaling, insensitive to lighting changes

A location (x,y) in the source image w/ an associated orientation, scale, & is "visually distinct" from surroundings.

$$p_i = (x_i', y_i', \rho_i', \theta_i)$$

Keypoint Detection

1. Build a Gaussian scaled-space, each image is smoothed by an additional factor of k

   $I_{k^s\sigma} = I * G_{k^s\sigma}$, $I_{k^{s-1}\sigma} = I * G_{k^{s-1}\sigma}$, ..., $I_{k\sigma} = I * G_{k\sigma}$, $I_\sigma = I * G_\sigma$

   Octave a sequence of Gaussian filtered images representing a cummulative doubling of Gaussian parameter σ

   $k^s\sigma = 2\sigma$, each octave contains $s+1$ images (in practice, $s=3$, $\sigma = 1.6$, 4 octaves)

   Images in next octave are subsampled, stored at 1/2 resolution of previous

2. Build DoG pyramid

   $I * (G_{k^{s-1}\sigma} - G_{k^s\sigma}) = I * D_{k^{s-1}\sigma}$, $I * D_{k^{s-2}\sigma}$, ..., $I * D_{k\sigma}$, $I * D_\sigma$

   Recall 2D difference of Gaussians (DoG) is a scaled Laplacian: $G_\sigma - G_{k\sigma} \simeq (1-k)\sigma^2 \frac{\partial^2 G_\sigma}{\partial x^2} \Rightarrow \frac{\partial G_\sigma}{\partial \sigma} = \sigma \frac{\partial^2 G_\sigma}{\partial x^2}$

3. Locate extrema of DoG pyramid (x_i, y_i, ρ_i)

   For each pixel $(x,y) \in D_\rho$, check if $D_\rho(x,y)$ is $>$ (or $<$) all its neighbours in current scale & adjacent scales

   An extremum detected at $D_\rho(x,y)$ defines a keypoint (x,y,ρ)

4. Refine location of DoG extrema by localizing extrema w/ sub-pixel & sub-scale accuracy $(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

   Fit 2nd degree poly $D(x,y,\rho)$ to DoG samples in neighbourhood of $D_\rho(x,y)$ using least squares fitting.

   Return (x', y', ρ') where $D(x', y', \rho')$ has an extremum (partials wrt x, y, ρ are 0)

   Discrete pixels & scales provide approximate estimate of keypoint location $\Rightarrow$ improve accuracy by poly fitting

5. Prune DoG extrema weak/edge-like extrema will likely lead to unstable/poorly localizable keypoints

   Reject extrema under a threshold & poorly localizable extrema (using eigenvalues of Hessian of D)

   ↳ get rid of saddles?

6. Assign orientation to keypoint (x_i', y_i', ρ_i')

   Compute ∇ magnitude & orientation in neighbourhood of (x_i', y_i') in Gaussian pyramid image closest to scale ρ_i'

   Compute histogram of gradient orientations (weighted by distance?)

   Assign the orientation θ_i w/ most votes.

Keypoint Descriptor

1. Compute gradients in 16 × 16 pixel patch of image I_p centered at (x_i, y_i) 
2. Compute gradient orientation relative to keypoint orientation
3. Compute orientation histogram of each 4 × 4 pixel block, 8 bins w/ 45°, length represents votes

   Total vote of (x, y) $G_{\frac{\rho}{2}}(\|(x - x_i, y - y_i)\|)\,\|\nabla I * G_{\sigma_i}(x, y)\|$

   List each of the votes in a vector (128, b/c 8 different bins, 16 4×4 blocks)
4. Given f_i, normalize $\left(f_i \leftarrow \frac{f_i}{\|f_i\|}\right)$

   $f_i = [\leftarrow \ \nwarrow \ \uparrow \ \nearrow \ \rightarrow \ \searrow \ \downarrow \ \swarrow, \ldots]$
5. Clamp all elements of f_i at 0.2 to give less weight to very large gradient magnitudes
6. Re-normalize

SIFT Applications

Panoramic mosaic using Autostitch, identify where to stitch

SIFT on Mars, piece together rover images

RANdom SAmple Consensus

my brain is being ransaced right now holy shit

⓵ stitching images, we need to robustly compute transformation in presence of wrong matches (outliers)

If # outliers < 50%, then RANSAC is extremely useful. (fuck around & find out)

p: fraction of inliners, t: inliner threshold, p_s: success probability, n: polynomial degree (for polynomials)

Minimal sample set the smallest set that yields a unique model

↳ line, n = 1, can be determined by 2 points

1. Randomly draw a minimal sample set
2. Fit (unique) model to those samples
3. Count inliners that fit the model within a measure of error ε
4. Repeat steps 1-3 at most k times
5. Choose largest inliner set & fit model w/ least squared error

Goal choose k st. a minimal sample set that contains no outliers will be found w/ probability p_s within k iterations

Stupid probability later (bin), $k \log(1 - p^{(n+1)}) = \log(1 - p_s) \Rightarrow k \geq \frac{\log(1 - p_s)}{\log(1 - p^{(n+1)})}$

Robust Homography Estimations

Goal is to estimate a homography matrix H given N point correspondences across two images.

Inliners: correctly-established SIFT correspondences, Outliers: incorrect, N: at least 4 correspondences

Given set of correspondences $(x_1, y_1) \rightarrow (x_1', y_1'), \ldots, (x_n, y_n) \rightarrow (x_n', y_n')$, run RANSAC.

Randomly draw 4 correspondences, compute H, count inliners (within an error ε bound), rinse & repeat, pick best one

Once an approx. H & inlinear correspondences are identified, use all of them (over-determined LS) for a better H

idk if this will be on but just in case

Gaussian Pyramid Representation

TLDR, Gaussian then downsample

Goal decompose images into information at multiple scales to extract features/structures of interest, attenuate noise

Some applications include efficient processing, image blending, image enhancement

Input: image I of size $(2^N+1) \times (2^N+1)$, output: N images $g_0, \dots, g_{N-1}$, where g_ℓ has size $(2^{N-\ell}+1)^2$

1. **Smoothing** smooth image w/ sequence of smoothing filters, each of which has twice the previous (like SIFT)

   With g_0 being the original photo, we start off w/

$$\hat{g}_1(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) \cdot g_0(i-m, j-n)$$

5×5 filter

$$\Longrightarrow \quad \hat{g}_1 = w * g_0$$

$$\hat{g}_2 = w * \hat{g}_1 = w * (w * g_0)$$

can be thought of as $h = w * w$, radius twice that of w

I'm too lazy to write properties

$$\hat{g}_3 = (w * w * w) * g_0$$

radius 4 times that of w

2. **Downsampling** reduce image size by 1/2 after each smoothing

Smoothing + downsampling are combined into a single REDUCE function

$$g_{\ell+1} = \text{REDUCE}(g_\ell)$$

$$g_{\ell+1}(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) \cdot g_\ell(2i-m, 2j-n)$$

downsample

smooth

Laplacian pyramid can go die in a hole fuck this shit im out.

IMAGE GEOMETRY

Homogeneous Coordinates

2D points can be represented using homogeneous coordinates

$$\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

To go back, $\begin{bmatrix} x \\ y \\ z \end{bmatrix} \longrightarrow \begin{bmatrix} x/z \\ y/z \end{bmatrix}$

For a homogeneous representation of p, it can be represented by any 3D vector $\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix}$, $\lambda \neq 0$

Two homogeneous vectors a v, v' are equal if they differ by a scalar ($v \cong \lambda v'$)
They are the same point.

Points at infinity have their last coordinate as 0 $\rightarrow [a \;\; b \;\; 0]^T$ finite representation

Lines can be represented in homogeneous coordinates

$$ax + by + c = 0 \longrightarrow \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \longrightarrow l^T \cdot p = 0$$

standard form; homogeneous line coordinates; line equation

If given two points p_1, p_2 that a line passes through, $l = p_1 \times p_2$

Given two lines l_1, l_2, their intersection point is $p = l_1 \times l_2$ (If 3rd coordinate is 0, lines are parallel)

Affine Transformations

Affine transformations are linear mappings that preserve parallelism.

↳

Stretch along x & y $\begin{bmatrix} s & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & 1 \end{bmatrix}$

rotation by θ° $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

shear along x & y $\begin{bmatrix} 1 & g & 0 \\ h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

general format $\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & g \end{bmatrix}$

these two must be 0 to maintain parallelism

If there is at least one non-zero, no longer parallel after transformation

Homography

eh.

same as affine, but $\begin{bmatrix} a & b & c \\ d & e & f \\ l & m & g \end{bmatrix}$

at least one non-zero

Image Warping

TLDR, map from dest to source to avoid "holes" or "overlaps" —— backward mapping

for r in dest_rows, for c in dest_cols:

get dest pixel's (x, y), convert to homogeneous, apply inverse homography, convert to Euclidean coords to access source pixel, write source to dest.

Perspective Viewing

Pinholes prevent cameras & eyes from seeing blurry images by filtering light.

aka. camera obscura

If aperture becomes too large, image becomes blurry.

The focal length f is the distance between image plane & aperture plane.

Let subscript i denote the projected, subscript 0 denote original

$$\frac{y_i}{y_0} = \frac{f}{z_0} \longrightarrow y_i = \frac{f}{z_0} y_0 \quad , \quad x_i = \frac{f}{z_0} x_0 \quad , \quad \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \frac{f}{z_0} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

z_0 = distance between source & aperture plane

Photos can be homography-aligned if: images are perspective images of a plane

perspective images of any scene are taken from same center of projection

cannot: non-planar surface

images have non-linear distortion

vanishing point stuff — Projection of parallel lines converges to a vanishing point