CS420 Image Understanding

© Tingfeng Xia

Fall Term, 2020

Information

- Syllabus can be found through this link.
- There will be office hours every weekday. :D

Contents

1 Linear Filter

1.1 Fourier Transform Overview

1.1.1 Sinusoidal Waves

Consider a sinusoidal wave, it has a general form of

$$A\cos\left(\omega t-\phi\right) \tag{1.1}$$

where A is the amplitude, $\omega=2\pi f$, where f is the frequency, and ϕ is called phase. Phase describes the horizontal shifting for the wave away from the standard position.

1.1.2 Vector Forms

Consider a vector $\mathbf{v}=(3,2)^{\top}$, we write it as

$$\mathbf{v}=3\mathbf{i}+2\mathbf{j}=(\mathbf{v}\cdot\mathbf{i})\mathbf{i}+(\mathbf{v}\cdot\mathbf{j})\mathbf{j} \tag{1.2}$$

and we call $\{\mathbf{i},\mathbf{j}\}$ an orthonormal basis[1.1].

1.1.3 Inner Product on Functions Space

Function space is an inner product space, and we can define, for a set of parametric function, on an interval of $[a,b]$ that

$$\langle f(t),g(t)\rangle=\int_a^b f(t)g(t)\,\mathrm{d}t \tag{1.3}$$

1.1.4 General Form (Periodic Function) - Fourier Series

For a function $f(t)$ that is periodic with period of T, the general form can ben written as

$$f(t)=\frac{a_0}{2}+\sum_{k=1}^{\infty}a_k\cos\left(\frac{2\pi kt}{T}\right)+\sum_{k=1}^{\infty}b_k\sin\left(\frac{2\pi kt}{T}\right) \tag{1.4}$$

$$=\frac{a_0}{2}\sum_{k=1}^{\infty}A_k\cos\left(\frac{2\pi kt}{T}-\phi_k\right) \tag{1.5}$$

where in the second form, we merged all the sin and cos terms with the introduction of a phase term. In particular, this now takes the same form as discussed in Section 1.1.1.

[1.1] $\|\mathbf{i}\|=\|\mathbf{j}\|=1\wedge\mathbf{i}\cdot\mathbf{j}=0$

1.1.5 General Form

The general form also takes into consideration of non-periodic functions. In such case, we need to change the summation into a integral. For any function $f(t)$, we have

$$F(\omega) = \int f(t)e^{-i\omega t}\mathrm{d}t \tag{1.6}$$

1.2 Image Representation

1.2.1 Image

Image is a matrix with integer values. The matrix would typically be denoted as I, and $I_{i,j}$ is called the ***intensity***. For each pixel, we usually represent it use an unsigned 8 bit unsigned integer and thus have range $2^0 = 0$ to $2^8 - 1 = 255$. For High Dynamic Range (HDR) images, they will be represented with 16 bit unsigned integer. Also there are cases that we (linearly) normalize the values by squashing them into $[0, 1]$.

1.2.2 Image Coordinates

Image coordinates start from the top left. For a coordinate (i, j), i specifies that it is in the i-th row, and j specifies column. Also worth noticing that the most upper left pixel has coordinates $(1, 1)$.

1.2.3 Coloured Images

In grey scale images mentioned in the previous two parts, for a image of size $m \times n$, we have a matrix of $m \times n$ 8-bit unsigned integers. Now with the introduction of colours, we will have a tensor of $m \times n \times 3$ 8-bit ints, corresponding to three colour channels. By convention, they usually goes in the order of $R \to G \to B$. For example, $I(2, 3, 1)$ means the intensity of ***red*** channel of the image at location row 2 and column 3.

1.2.4 Image Transformations

For simplicity, we start with grey scale images. We can view any image as a function $f : \mathbb{R}^2 \rightharpoonup \mathbb{Z}_{0-255}$, and this enables us to transform images. An easy example would be to increase the brightness of the image, which we can achieve so with

$$J(i, j) = \min\{I(i, j) + \text{amount}, 255\} \tag{1.7}$$

capping the max intensity at 255. Importantly, we can so some interesting operations by treating images as functions. Namely, correlation and convolution.

1.3 Noise Reduction

1.3.1 1-D Example

Consider a signal, which is a real to real function, and our goal is to smooth the function by imposing human knowledge that the signal should be smooth and should not contain too many jitter. We can have

- **Moving Average Filter**, which is $[1, ..., 1]/n$, or
- **Non-uniform Weights**, for example $[1, 4, 6, 4, 1]/16$

1.3.2 2-D Case

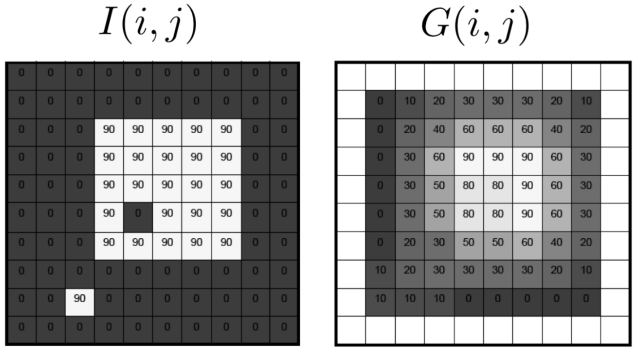

Figure 1: Example of using moving average to smooth out an image, *assuming no padding*.

Much similar to the 1-D case mentioned above, we have our choice of whether to choose an uniform filter or not.

- In the case of uniform (aka moving average), we choose (example of 3×3 filter)

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} /9 \tag{1.8}$$

  Figure 1 is an example of using moving average to smooth out an image, *assuming no padding*. Notice that the sharp boundaries that we used to have between dark and white is now smooth. Also, the isolated pixels ($(6, 5)$ and $(9, 3)$) are now blended in.

  De-noising is usually an important first step (preprocessing) in any image task

- In case of non-uniform, we can again choose a gaussian like filter, such as

$$\begin{bmatrix} 1 & 4 & 1 \\ 4 & 10 & 4 \\ 1 & 4 & 1 \end{bmatrix} /30 \tag{1.9}$$

1.4 Correlation Defined

1.4.1 General Moving Average

In the general case, our filter could be any size. In particular, it needs to be of size square of an odd number. Then, the moving average becomes

$$G(i,j) = \frac{1}{(2k+1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} I(i+u, j+v) \tag{1.10}$$

1.4.2 General Filtering

If we apply some filter (i.e., not just a average) then we can use

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v) \cdot I(i+u, j+v) \tag{1.11}$$

where $F(\cdot,\cdot) : \mathbb{R}^2 \to \mathbb{R}$ is the called ***kernel*** or ***mask*** or ***filter*** such that $\sum_{u'} \sum_{v'} F(u,v) = 1$. The elements of the filter is called ***filter coefficients***. Notice that if we take ($|V|, |U|$ here denotes the max values that v, u can take)

$$F(u,v) = \frac{1}{(2|U|+1)^2} = \frac{1}{(2|V|+1)^2} \tag{1.12}$$

then Equation 1.11 just collapses to Equation 1.10.

1.4.3 Notation

The Filtering operation defined above is called *correlation*, denoted as

$$G = F \otimes I \tag{1.13}$$

where F is our filter / kernel / mask, and I is the original image.

Notice that filter is also an image, so $\otimes$ essentially takes two images as input and outputs one image.

1.4.4 Correlation - Vector Form

Define

- $\mathbf{f} = F(:)$, writing the matrix into a vector.
- $T_{ij} = I(i-k : i+k, j-k : j+k)$, the part of image covered by the filter around original image at coordinates (i,j)
- $\mathbf{t}_{ij} = T_{ij}(:)$ putting the part of image selected in previous step into a vector.

then,

$$G(i,j) = \langle \mathbf{f}, \mathbf{t}_{ij} \rangle = \|\mathbf{f}\| \, \|\mathbf{t}_{ij}\| \cos\theta \tag{1.14}$$

which converts two for loops into one inner product. This is much faster to compute as far as codes are concerned.

TODO: above we defined correlation for one pixel as a vector operation, can we define the entire correlation into matrix form

1.4.5 Normalized Cross-correlation

In the task of finding Waldo, we wish to get a score of whether or not a patch of image looks like Waldo. In particular, we want this score to be the highest for the patch with Waldo, but not a very bright patch without Waldo. In an hope to achieve this goal, we can use normalized cross correlation: (utilizing the vector forms in the previous section)

$$G(i,j) = \frac{\mathbf{f}^\top \mathbf{t}_{ij}}{\|\mathbf{f}\| \, \|\mathbf{t}_{\mathrm{ij}}\|} = \cos\theta \tag{1.15}$$

where θ is the angle between vectors $\mathbf{f}$ and $\mathbf{t}_{ij}$

1.5 Boundary Effects

Assume we have image size of $m \times n$, and filter size of $k \times k$. Referring to `cv2.filter2d` in OpenCV and `filter2(F, I, SHAPE)`, we have the following cases:

- `shape = 'full'` output size is bigger than the image; infinite padding include all reasonable values. Output should have size $(m + 2k - 2) \times (n + 2k - 2)$
- `shape = 'same'` output size is same as I; padding such that output size is equal to input size.
- `shape = 'valid'` output size is smaller than the image; no zero padding; output should be size $(n - k + 1) \times (m - k + 1)$

1.6 Smoothing

1.6.1 Uniform Smoothing

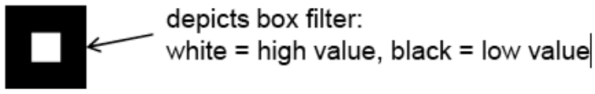

Figure 2: Box Filter

The box filter depicted in Figure 2 is the exact same filter if we only keep the white part, i.e. the 1 entries. As the size of the box filter increases, the end result gets more and more blurry.

1.6.2 Isotropic Gaussian Filter

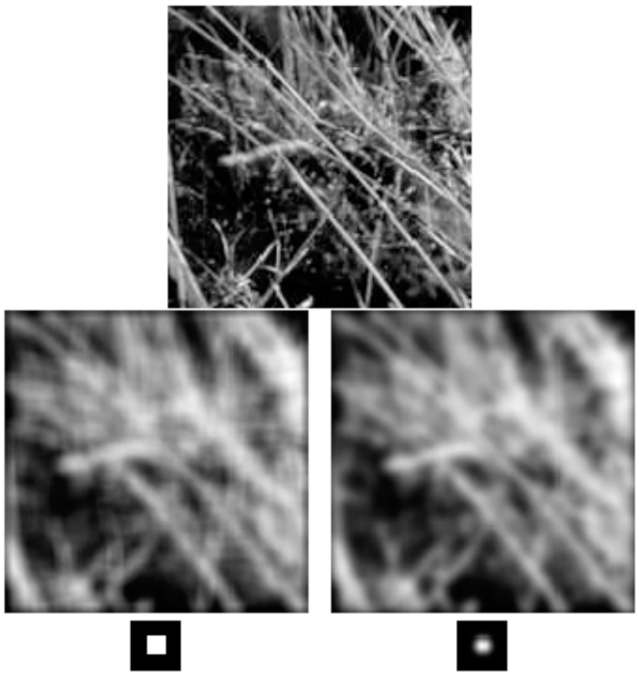

Figure 3: Comparison of filtered result using uniform filter (bottom left) and gaussian filter (bottom right)

Recall that the gaussian probability distribution is defined as

$$\text{Gaussian}(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = (2\pi)^{-\frac{k}{2}} \det(\Sigma)^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{1.16}$$

and we can mimic this to develop a filter that has entries mimicking values taken by the gaussian pdf. These filters produce results much nice than averaging in terms of smoothing, as we can see in Figure 3.

Specification The Gaussian Filter G is parametrized by two parameters $\Sigma = \sigma I$ (isotropic) and $\boldsymbol{\mu}$. In application, $\boldsymbol{\mu}$ doesn't matter, we always want to make sure that the peak of the gaussian pdf corresponds to the centre pixel of the filter. The size of the filter depends

on our choice of Σ, e.g. it doesn't make much sense if our kernel includes values more than 2σ's away. We also need to normalize all the taken values again, to make sure the filter we chose sums up to one!

1.6.3 Non-isotropic Gaussian Filter

In the most general case, Gaussian can be non-isotropic, meaning that its variance-covariance matrix ***is not*** of form σI

1.7 Convolution

The Convolution operation is defined as

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v) \cdot I(i-u, j-v) \tag{1.17}$$

Notice that this is exactly the same as Correlation defined in Equation 1.11 except that we are flipping the filter in both dimensions (bottom to top, right to left).

In the case of Gaussian / box filters, since the filter will be symmetric about both horizontal and vertical axis, $F * I = F \otimes I$.

1.7.1 Properties of Convolution

Convolution is a Linear Operation, meaning that if f, g and h are three convolution operators, and $\lambda \in \mathbb{R}$ then

- ***Commutative***: $f * g = g * f$
- ***Associative***: $f * (g * h) = (f * g) * h$
- ***Distributive***: $f * (g + h) = f * g + f * h$
- ***Assoc. with scalar multiplier***: $\lambda \cdot (f * g) = (\lambda \cdot f) * g$

1.7.2 Convolution w/ Fourier Transforms

The Fourier transform of two convolved images is the inner product of their individual Fourier Transforms, i.e.

$$\mathcal{F}(f * g) = \langle \mathcal{F}(f), \mathcal{F}(g) \rangle \tag{1.18}$$

Implications The computational complexity of Fourier Transform is much lower than that of convolution. Also notice that inner products are fast to compute.

Confirm and finish this !

1.8 Separable Filters

For a $K \times K$ sized filter / kernel, the process of performing a convolution requires K^2 operations per pixel, summing up to a total of #pixels $\times K^2$ for an entire image. In many cases, though not all, we can speed this process up by (1) performing a 1-D horizontal convolution followed by (2) a 1D vertical convolution, requiring only $2K$ operations! When we can do this trick, we call the kernel "separable". And a filter is separable iff it is the outer product of two vectors (each a 1D filter):

$$\exists? \, \mathbf{v}, \mathbf{h} \in \mathbb{R}^K, s.t. \; F = \mathbf{v}\mathbf{h}^\top \tag{1.19}$$

1.8.1 Isotropic Gaussian as Separable Filters

One famous example of separable filter that we are already familiar with is the Gaussian Filter, assuming isotropic variance. In such case the density breaks down into

$$\text{Gaussian}(x, y; \boldsymbol{\mu} = \mathbf{0}, \Sigma = \sigma I) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{x^2+y^2}{\sigma^2}\right\} \tag{1.20}$$

$$= \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{x^2}{\sigma^2}\right\}\right] \cdot \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{y^2}{\sigma^2}\right\}\right] \tag{1.21}$$

Such factorization of gaussian pdf indicates us that we should have two 1-D filters that are 1-D gaussian each.

1.8.2 Moving Average as Separable Filters

The naïve moving average filter that we encountered earlier is also separable, in which case

$$F = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix} / K^2 = \begin{bmatrix} 1/K & \dots & 1/K \end{bmatrix}^\top \begin{bmatrix} 1/K & \dots & 1/K \end{bmatrix} \tag{1.22}$$

1.8.3 Edge Detector Kernel as Separable Filters

The edge detector

Come back: left or right edge detector?

$$F = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} /8; \quad \mathbf{v} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} /2; \quad F = \mathbf{v}^\top \mathbf{v} \tag{1.23}$$

1.8.4 Separable-ness of Filters

A systematic way of checking if a kernel is separable if by looking at the singular value decomposition of the filter. ***If only one singular value is non-zero, the it is sepa-***

rable

$$F = \mathbf{U}\Sigma\mathbf{V}^\top = \sum_i \sigma_i u_i v_i^\top \tag{1.24}$$

with $\Sigma = \text{diag}(\sigma_i)$. Then, we can get the vertical and horizontal filter through

$$F_{\text{vertical}} \leftarrow \sqrt{\sigma_1}\mathbf{u}_1 \qquad F_{\text{horizontal}} \leftarrow \sqrt{\sigma_1}\mathbf{v}_1^\top \tag{1.25}$$

2 Edge Detection

2.1 Characterization of Edges

2.1.1 Insights

- Edge detection involves mapping image to a set of ***curves*** or ***line segments*** or ***contours***.
- Such representation is more compact than pixels. Notice that for a coloured (ordinary, not HDR) $m \times n$ image, we will need $m \times n \times 3 \times 8$ bits to store all the information. However, for edges $m \times n$ would suffice.
- They are particularly useful due to their invariance towards illumination - it thus helps computers see better. Aside: edges are also important for recognition for human.

2.1.2 Origin of Edges

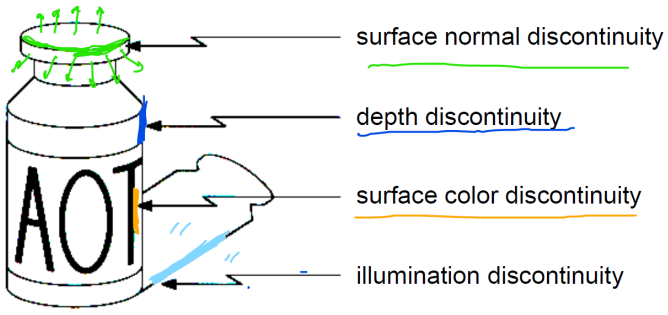

Figure 4: Four origins of edges

Figure 4 illustrates the four types of edges that can occur.

- ***Surface Normal Discontinuity*** is where surface normals change direction abruptly.

- ***Depth Discontinuity*** is caused by depth discrepancy between two objects from the angle of the viewer. For example here in Figure 4, the background and the bottle causes a depth discontinuity.
- ***Surface Colour Discontinuity*** is for example the edge of black text T on a white bottle.
- ***Illumination Discontinuity*** is when there is a shadow causing difference in light.

2.1.3 Characterizing Edges

Definition An edge is a place of rapid change in image intensity function. The means that at places where edges occur, the intensity function should be steep, and the first derivative of the intensity at that position should correspond to extrema.

2.2 Convolution as Derivative - Measure of Rapid Change

Consider an image $f(x, y)$ defined for $x \in \mathbb{Z}^{\geq 1, \leq m}, y \in \mathbb{Z}^{\geq 1, \leq n}$, how can we differentiate this digital image given that it is not continuous? The answer is we take the first order forward discrete derivative (finite difference), i.e.

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f[x+1,y] - f[x,y]}{1} \tag{2.1}$$

and

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f[x,y+1] - f[x,y]}{1} \tag{2.2}$$

Correlation Filter Clearly we can implement the above as kernels,

$$H_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad H_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.3}$$

then, we have

$$[f \otimes H_x]_{(i,j)} = \left.\frac{\partial f(x,y)}{\partial x}\right|_{(i,j)} \quad \text{and} \quad [f \otimes H_y]_{(i,j)} = \left.\frac{\partial f(x,y)}{\partial y}\right|_{(i,j)} \tag{2.4}$$

2.2.1 Canonical Finite Difference Filters

The Prewitt Kernel is more symmetric, and averages each pixel from neighbouring pixels only. Also this filter applies a tiny blurring on the direction that it is not detecting edge. (M_x is blurring for vertical direction.)

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.5}$$

Sobel Filter is more common. Same as Prewitt, in the direction of that the kernel is not detecting edges, it applies a bit blurring effect. *However, in Sobel, the blurring is a Gaussian blur.*

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.6}$$

Roberts Kernel detects diagonal edges;

$$M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{and} \quad M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.7}$$

2.3 Image Gradient

2.3.1 Gradient Defined

The gradient of an image $f : \mathbb{R}^2 \to \mathbb{R}$ is defined, exactly the same as usual, as

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \tag{2.8}$$

2.3.2 Edge Direction

The gradient always points in the direction of most rapid change in intensity. This means that if

$$\nabla f = \left[\frac{\partial f}{\partial x} \neq 0, \to 0 \right] \tag{2.9}$$

then that position corresponds to a vertical edge. If

$$\nabla f = \left[\to 0, \frac{\partial f}{\partial y} \neq 0 \right] \tag{2.10}$$

when we are at a horizontal edge. At a slanted edge, we will get

$$\nabla f = \left[\frac{\partial f}{\partial x} \neq 0, \frac{\partial f}{\partial y} \neq 0 \right] \tag{2.11}$$

2.3.3 Gradient Direction

The gradient direction (i.e. orientation of edge normal) is given by

$$\tan\theta = \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \implies \theta = \arctan\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right) \tag{2.12}$$

2.3.4 Edge Strength

The edge strength is given by the L_2 norm of the gradient vector:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{2.13}$$

2.4 Effects of Noise

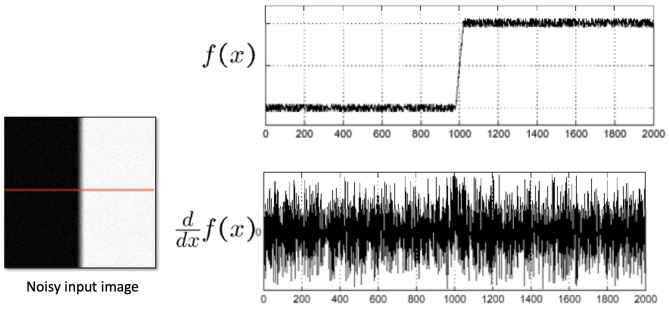

Figure 5: Illustration of noisy input problem.

As shown in Figure 5, when we have edges that are not sharp, the image is noisy and thus cause the derivative messy.

2.4.1 Overcoming Noisiness

The solution to the problem is easy. We simply first smooth the input signal and then look for edges. I.e., for an input image f, and noise reduction filter (e.g. Gaussian) h, we find extremum in $\frac{\partial}{\partial x}(h * f)$

2.4.2 Faster Method Through Conv / Corre

We know that correlation is associative, i.e. $f*(g*h) = (f*g)*h$. We can use this property to speed up our method of overcoming noisiness mentioned in Section 2.4.1. Consider an image $I \in M_{m\times n}(\mathbb{R})$, smoothing filter $G \in M_{k\times k}(\mathbb{R})$, and x-derivative kernel $F \in M_{k\times k}(\mathbb{R})$. Then,

$$\frac{\partial}{\partial x}(G * I) = F * (G * I)) \tag{2.14}$$

$$= (F * G) * I \tag{2.15}$$

and since $k \ll m, n$, Equation 2.15 i s much faster to compute.

2.4.3 Generalization: Derivative Theorem of Convolution

For a image f, with filter h, we have

$$\frac{\partial}{\partial x}(h * f) = \left(\frac{\partial h}{\partial x}\right) * f = h * \left(\frac{\partial f}{\partial x}\right) \tag{2.16}$$

i.e. rather than convolving the image with the filter and then take the derivative, we first get the derivative of the filter and convolve the result with the image.

This saves us one operation.

2.4.4 Remark: on Gaussian Derivative Filters' Parameter

We know that if we apply the derivative of gaussian as a filter to an image, it finds the edges on the smoothened version of the image. However, the detected structures differ depending on the Gaussian's std. deviation chosen.

- If we have a large σ, then the filter detects edges of larger scale
- else if we have a small σ, we will be detecting finer structures.

2.5 Canny's Edge Detector

2.5.1 Procedure

1. Filter images with derivative of gaussian (horizontal and vertical directions)
2. Find magnitude and orientation of gradient
3. Non-maximum suppression
4. Linking and thresholding (hysteresis)
    - Define two thresholds: low and high
    - Use the high threshold to start edge curves and the low threshold to continue them

Parameters There are three parameters (hyper-parameters) that we need to fix for the algorithm, namely scale of Gaussian in step 1, and low / high thresholds in step 4. There are no magical way in tuning them, so it requires quite a lot of experimentation.

2.5.2 Non-Maximum Suppression (Thick Edges Problem)

- check if pixel is local maximum along gradient direction?
- if yes, take it; otherwise neglect it.

i.e. in an image f we take (i, j) if in a local area

$$\forall (i', j'), I(i, j) > I(i', j') \tag{2.17}$$

then keep $\|\nabla f\| \leftarrow \|\nabla f\|$ otherwise $\|\nabla f\| \leftarrow 0$

2.5.3 Hysteresis Thresholding (Discontinuous Edges Problem)

- Filter at high threshold, getting strong edges. Call this result S
- Filter at low threshold, getting weak edges, call this result W
- Along directions where edges develop in S, if there is an edge in W at the same spot, then we adopt it. Continue until a point where norm of gradient is below the low threshold.

2.6 Laplacian of Gaussians: Another Approach to Edge Detection

Consider (noisy) image f, and a Gaussian filter h. We have the laplacian of h, as $\frac{\partial^2}{\partial x^2} h$, and then we calculate

$$\left(\frac{\partial^2}{\partial x^2} h \right) * f \tag{2.18}$$

where zero-crossings in the resulting graph correspond to edges.

2.7 Auxiliaries

2.7.1 Connection: Sobel Filter and Gaussian Blur

We already know that

$$F * (G * I) = (F * G) * I \tag{2.19}$$

and suppose

$$F = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} / 14 \tag{2.20}$$

then if we use the convolution sequence on the right, we will get

$$\begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -2 & -4 & 0 & 4 & 2 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix} \in M_{3 \times 5}(\mathbb{R}) \tag{2.21}$$

as an intermediate result. But this non-square result is not "nice". So more commonly we use

$$G = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^\top / 4 \tag{2.22}$$

in which case

$$F * G = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.23}$$

and we notice that the result is exact the same as Sobel Filter we mentioned earlier.

3 Image Pyramids

In Linear Filter section, we talked about the technique of using correlation / convolution to aid us find Waldo in a picture. However, that method relies on the filter and the matching portion in the image being the same size. What should we do when we have the template, but just a thumbnail image of the filter?

3.1 Image Sub-Sampling

The goal is very simple, how do we make a smaller sized image out of an original image? e.g. 1/4 sized, 1/8 sized...

3.1.1 Naïve Solution

Simply remove some columns and rows based on a predetermined rule, e.g. remove one in every two row / column. In our example, the remaining pixel makes a 1/2 sized image. We can do this again and again, creating an $1/2^n$ sized image.

Pitfall - Natural Image This very simple solution creates images that look very crufty (containing very sharp noises).

Pitfall - Synthetic Image In a computer synthesized image where there is a box, with line width of 1px. Now I wish to resize the image by a factor of 2 by taking away every other column and every other row (1st, 3rd, 5th, etc). But then, if any of the column / row of pixel in the image is the same as the "every other row" discarded, then we have problem - our box will have missing sides. And this is not a rare event.

3.1.2 Aliasing

Aliasing could occur when we sample from a source signal, and due to limitations in the sampling rate / density we get a completely different sampled signal output. We say that the sampling rate is not high enough to capture the amount of detail in the original signal.

This is commonly seen when we take a photograph for a digital screen and view it on a display with not high-enough resolution.

3.1.3 Nyquist Rate (Nyquist-Shannon Theorem)

Poor sampling creates aliasing. To do sampling right, we need to understand the structure of the input signal. The minimum sampling rate is called the Nyquist Rate[3.1]. Harry states that

- One should look at the frequencies of the signal, and
- find the highest frequency via Fourier Transform.
- To sample properly, you need to sample with at least twice that highest frequency.

In short, we need

$$\text{Sampling Freq.} > 2 \times \text{Max Freq. Component of Original Signal} \tag{3.1}$$

3.1.4 Gaussian Pre-Filtering

We now see that high frequency signals are "some-what" the problem in signal sub sampling. In image, this typically correspond to sharp edges. The solution to this problem is to use a Gaussian Filter to help us do a pre-filtering. The blurring helps aggregate, at each resulting pixel positions, information from pixels around, making the image smoother [3.2]

Theory Behind Recall that Gaussians are exponential, so we can write the pdf $f(x)$ up to a scaler multiplicand difference

$$f(x) = \exp\{-ax^2\} \propto \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{\frac{-x^2}{2\sigma^2}\right\} \tag{3.2}$$

where $a = 1/2\sigma^2$. Then, the Fourier Transformation is

$$F(f(x)) = \int_{-\infty}^{+\infty} e^{-ax^2} e^{-2\pi ikx} \mathrm{d}\, x = f(f(x))(k) \tag{3.3}$$

$$= \int_{-\infty}^{+\infty} e^{-ax^2} \left(\cos(-2\pi ikx) + i\sin(-2\pi ikx)\right) \mathrm{d}\, x \tag{3.4}$$

$$= \int_{-\infty}^{+\infty} e^{-ax^2} \cos(-2\pi ikx) \mathrm{d}\, x \tag{3.5}$$

$$= \sqrt{\frac{\pi}{a}} \exp\left\{\frac{-\pi^2 k^2}{a}\right\} \tag{3.6}$$

[3.1] nameed after Sir Harry Nyquist. https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

[3.2] I think this can also be explained by Statistical Multiplexing.

3.1.5 Image Pyramids

A sequence of images created with Gaussian blurring and downsampling is called a Gaussian Pyramid. We represent a $N \times N$ sized image as a pyramid of 1×1, 2×2, ..., $2^k \times 2^k$ images, assuming $N = 2^k$.

In CV, this is called a *mip map*

3.2 Image Up Sampling

Now we consider the opposite question: if an image is too small, how can we make it 10 times larger?

3.2.1 Naïve Solution

We can repeat each row and column 10 times and create a "quasi" sized up image.

Pitfall We notice that this method of sizing up just made the size of the image larger, but it does so by creating blobs of pixels representing the same pixel. This result is not desirable.

3.2.2 1-D Signal Linear Interpolation

We can make a linear interpolation between neighbouring discrete points in the input signal to create a better up sampling solution. Suppose input is $F(x)$, and between points x_1 and x_2, we wish to sample a x. Then,

$$x = \alpha x_1 + (1 - \alpha)x_2, \alpha \in (0, 1) \tag{3.7}$$

and the estimated signal strength (by linear interpolation) is

$$\hat{F}(x) = \alpha F(x_1) + (1 - \alpha)F(x_2), \alpha \in (0, 1) \tag{3.8}$$

Rewriting and substituting gives us the final formula of

$$\hat{F}(x) = \frac{x_2 - x}{x_2 - x_1} F(x_1) + \frac{x - x_1}{x_2 - x_1} F(x_2) \tag{3.9}$$

3.2.3 1-D Linear Interpolation Via Convolution

Consider 1-D input signal F, and we first "expand" it (filling gaps between signal points with 0s). We call this expanded version G'. Then, to get final result G, it suffices to compute

$$G = h * G' \tag{3.10}$$

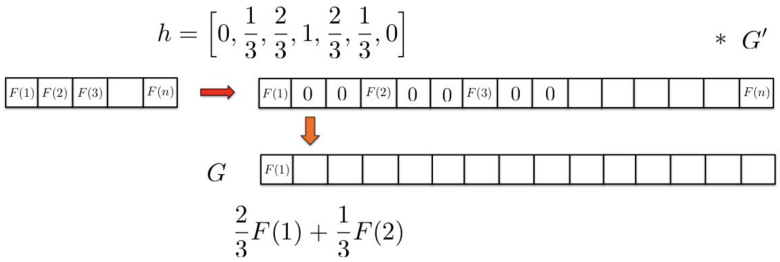

Figure 6: Worked example of up sampling with linear interpolation via convolution. h shown is the convolution filter, G' is the expanded input, and G is final result.

Example ($d = 3$) We can work out a h in a simple case of $d = 3$, as shown in Figure 6. But what about the general case? What should be my reconstruction filter h, such that $G = h * G'$?

General Case In general, we can find h using

$$h = \left[0, \frac{1}{d}, \ldots, \frac{d-1}{d}, 1, \frac{d-1}{d}, \ldots, \frac{1}{d}, 0\right] \tag{3.11}$$

where d is the up-sampling factor to help determine the reconstruction filter. Notice that the filter is symmetric.

3.2.4 1-D Non-Linear Interpolations

Previously we saw that we can use linear interpolation between points to help us reconstruct signals in up sampling. We can also use nonlinear functions as interpolation, for example

- $$\text{sinc}(x) \equiv \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin x}{x} & \text{otherwise} \end{cases} \tag{3.12}$$

  produces so called "ideal" reconstruction
- $\prod(x)$ produces nearest neighbour interpolation.
- $\Lambda(x)$ is our familiar linear interpolation
- Gauss(x) makes gaussian reconstruction

3.2.5 2-D Image Interpolation

4 Interest (Key) Point Detection

4.1 Review: Taylor Expansion

In general, if we have a function f that is $\mathcal{C}^{1}$[4.1] around a point a, than for all x in a neighbour hood of a,

$$f(x) \approx f(a) + f'(a)(x - a) \tag{4.1}$$

Let's call $x - a$ as y, and since the value is small, lets call a as Δy. Then,

$$f(y + \Delta y) = f(y) + f'(y)\Delta y \tag{4.2}$$

This also works for $\mathbb{R}^2 \to \mathbb{R}$ functions, and in particular works for images, in which

$$I(x + u, y + v) \approx I(x, y) + u\partial_x I(x, y) + v\partial_y I(x, y) \tag{4.3}$$

4.2 The Problem and Goal

Given two (or more) images of the same subject or very similar subjects, identify (at least some points in both images).

- We have to be able to run the detection procedure independently per image,
- we need to generate enough points to increase our chance of detecting matching pts,
- we should not generate too many key pts, or otherwise the algorithm will be really slow in checking matching pairs.

4.2.1 Naïve Point Choosing Criteria

We want to detect points that represent "corners" of objects in the image. This makes these patches more unique. If all the edges of corners are x and y axis aligned, then we can just choose points where both x and y directional partial derivatives are large. But this clearly is not a general solution.

4.3 Harris Corner Detector

4.3.1 Second Moment Matrix / Structure Tensor

Define the weighted sum (called weighted sum of squares difference)

$$E(u, v) = \sum_x \sum_y w(x, y) \left(I(x, y) - I(x + u, y + v)\right)^2 \tag{4.4}$$

[4.1] Will need $\mathcal{C}^\infty$ if we want entire taylor series, but we are just using first order approximation here.

Here $w(\cdot,\cdot)$ is called a window or weight function, $I(\cdot,\cdot)$ is called the intensity function. Our final goal is to make $E(u,v)$ large for any small (u,v). We can expand E,

$$E_{wssd}(u,v) = \sum_x \sum_y w(x,y)\left(I(x,y) - I(x+u,y+v)\right)^2 \tag{4.5}$$

$$\approx \sum_x \sum_y w(x,y)\left(I(x,y) - I(x,y) - u\partial xI(x,y) - v\partial_y I(x,y)\right)^2 \tag{4.6}$$

$$= \sum_x \sum_y w(x,y)(u^2\partial_x I\partial xI + 2uv\partial_x I\partial_y I + v^2\partial_y I\partial_y I) \tag{4.7}$$

$$= \sum_x \sum_y w(x,y)\begin{bmatrix}u & v\end{bmatrix}\begin{bmatrix}(\partial_x I)^2 & \partial_x I\partial_y I\\ \partial_x I\partial_y I & (\partial_y I)^2\end{bmatrix}\begin{bmatrix}u\\ v\end{bmatrix} \tag{4.8}$$

since the u, v extraneous variables, we can move them out,

$$E_{wssd}(u,v) = \begin{bmatrix}u & v\end{bmatrix}\underbrace{\left(\sum_x \sum_y w(x,y)\begin{bmatrix}(\partial_x I)^2 & \partial_x I\partial_y I\\ \partial_x I\partial_y I & (\partial_y I)^2\end{bmatrix}\right)}_{M\in M_{2\times 2}(\mathbb{R})}\begin{bmatrix}u\\ v\end{bmatrix} \tag{4.9}$$

4.3.2 Properties of Structure Tensor

For convinience, let $\dagger = \begin{bmatrix}(\partial_x I)^2 & \partial_x I\partial_y I\\ \partial_x I\partial_y I & (\partial_y I)^2\end{bmatrix}$

- Clearly W is symmetric and real, so it must have all real eigenvalues.
- $\det\dagger = 0$, so at least one of the eigenvalues of $\dagger$ is zero: $\lambda_1 = 0$ and $\lambda_2 \geq 0$. We say that $\dagger$ is a ***positive semi-definite matrix***.
- $\det M = \sum_x \sum_y w(x,y)\det(\dagger)$. The sum of positive semi-definite matrices is also positive semi-definite. Hence the eigenvalues of M are all ≥ 0. (there are two of them).
- Since it is symmetric and real, according to Spectral Theorem, this matrix is diagonalizable, and the rotation matrices are orthonormal.[4.2]We have

$$M = V\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}V^{-1} = V\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}V^{\top} \tag{4.10}$$

4.3.3 Weight Sum of Squared Difference Maximization

We saw above that we can diagonalize M, and if we expand this into E_{wssd}, then

$$E_{wssd}(u,v) = \begin{bmatrix}u & v\end{bmatrix}V\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}V^{-1}\begin{bmatrix}u\\ v\end{bmatrix} \tag{4.11}$$

[4.2]Orthonormal: $V^{\top}V = VV^{\top} = I$ and $V^{\top} = V^{-1}$

and this is big when both the two eigenvalues λ_1 and λ_2 are large.

4.3.4 Reverse Construction of Corners from WSSD Max Vals.

Above we saw the relationship between E_{wssd} and the eigenvalues, now we use this relationship to find out where in the image are the corners.

- If both λ's are small: Boring areas, not much change
- else if both λ's are big: corner!
- else if just one of them is big, then it is an edge.

4.3.5 Harris Corner Detector (Harris and Stephens, 88')

This is a heuristic function that is high when both eigenvalues are large, and low when at least one eigenvalue os small.

$$R = \lambda_0\lambda_1 - \alpha\left(\lambda_0 + \lambda_1\right)^2 = \det(M) - \alpha \cdot \text{trace}(M)^2 \tag{4.12}$$

where R is rotationally invariant and downweighs edge-like features where $\lambda_1 \gg \lambda_0$. α here is a hyper-parameter that is typically 0.04 to 0.06. If we plot R on axis of λ_1 and λ_2, then we will see

- when λ_1 and λ_2 both large, then R will be large
- when they are both small, R will be close to zero
- and when one of them small, the other large, R will be negative.

Note on Interchanging Notation Notice that

$$\det M = \det V \det \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \det V^{-1} \tag{4.13}$$

but since V is orthonormal, and importantly a rotation matrix, it has a determinant of 1. [4.3] Hence, $\det M = \lambda_1\lambda_2$.

Using Non-explicit Values We can notice that the formula is in terms of det and trace operators, rather than just of the eigenvalues. Although these two forms are same mathematically, they have different compute time. (you don not need to compute eigenvalues!)

4.3.6 Shi and Tomas, 94'

Shi and Tomas, 94' proposed using the smallest eigenvalue of M, i.e. $\lambda_0^{-1/2}$

[4.3] Recall that determinant can be defined as the change of volume of unit square / cube after transformation. Rotating doesn't change the volume of the shape, so determinant is 1.

4.3.7 Trigges, 04'

Suggested

$$\lambda_0 - \alpha\lambda_1 \tag{4.14}$$

also reduces the response at 1D edges, where aliasing errors sometimes inflate the smaller eigenvalue.

4.3.8 Brown et al, 05'

Suggested using the harmonic mean

$$\frac{\det(M)}{\text{trace}(M)} = \frac{\lambda_0\lambda_1}{\lambda_0 + \lambda_1} \tag{4.15}$$

5 Motion and Optical Flow

Perceiving, understanding and predicting motion is an important. Obviously, in this case the minimum amount of frame that we will need is two. Even the intelligent human brain can be fooled - we have all seen those static picture that seems to be moving. The question we wish explore in this section are

- Extract visual features[5.1] (corners, textured areas) and "track" them over multiple frames
- Recover image motion at each pixel from spatiotemporal image brightness variations. This is called ***Optical Flow***.

Applications

- We may wish to achieve video stabilization (removing shaking),
- analyzing moving objects in a (series of?) static frame

Feature Tracking Given two subsequent frames, say we want to estimate the point translation. Some assumptions that we make are (as proposed in the original Optical Flow paper; these assumptions are very strong)

- Brightness constancy: projection of the same point looks the same in every frame,
- small motion: points fo not move very far,
- spatial coherence: points (in a patch) move like their neighbours

[5.1] E.g. Edges, corners, textures, objects are all so called visual features

5.1 Brightness Constancy Constraint

Suppose we have video and at time t, pixel location (x, y) have brightness level $I(x, y, t)$. Then, if during time $t \to t+1$ it had a displacement (u, v), then it must be that

$$I(x, y, t) = I(x + u, y + v, t + 1) \tag{5.1}$$

We can expand RHS, using Taylor approximation, around (x, y, t),

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + u\partial_x I + v\partial_y I + 1\partial_t I \tag{5.2}$$

then we can reorder

$$I(x + u, y + v, t + 1) - I(x, y, t) = \approx u\partial_x I + v\partial_y I + 1\partial_t I \tag{5.3}$$

But since we assumed brightness constancy, the LHS of Equation 5.3 must be at least very close to zero. Then,

$$u\partial_x I + v\partial_y I + 1\partial_t I = 0 \tag{5.4}$$

$$\implies \nabla I \cdot \begin{bmatrix} u & v \end{bmatrix}^\top + \partial_t I = 0 \tag{5.5}$$

where ∇I is obtained by treating t as extraneous. We notice that our goal is to solve the image motion vector $\begin{bmatrix} u & v \end{bmatrix}^\top$, but we have only one equation for two unknown variables.

5.1.1 Pitfall

The component of the motion perpendicular to the gradient (parallel to edge) cannot be measured. If (u, v) satisfies the brightness constraint equation (Equation 5.5), then so does $(u + u', v + v')$ if $\nabla I \cdot \begin{bmatrix} u' & v' \end{bmatrix}^\top = 0$.[5.2]This tells us that the brightness Constancy Constraint will not help us recover motion along direction perpendicular to image gradient.

5.2 Ambiguities

5.2.1 The Aperture Problem

The aperture which was used to capture a motion can limit the way we interpret the motion. See a gif from wikipedia: https://en.wikipedia.org/wiki/Motion_perception#The_aperture_problem which illustrates this phenomenon. In naïve words, we cannot recover motion that was not seen.

[5.2]Since $\nabla I \cdot \begin{bmatrix} u+u' & v+v' \end{bmatrix}^\top = \nabla I \cdot \begin{bmatrix} u & v \end{bmatrix}^\top + \nabla I \cdot \begin{bmatrix} u' & v' \end{bmatrix}^\top$ but $\nabla I \cdot \begin{bmatrix} u' & v' \end{bmatrix}^\top$ could be potentially zero if the motion $\begin{bmatrix} u' & v' \end{bmatrix}$ is along direction perpendicular to gradient.

5.2.2 The Barber Pole Illusion

The revolving pole at entrance to barbers is really just revolving, but we perceive it as if it was moving upwards.

5.3 Spatial Coherent Constraint: Solving Ambiguities

The solution lies within how we can get more equations for a pixel. We need help from the Spatial Coherent Constraint. Assume that a pixel's neighbours have the same motion vector (u, v), and if we assume a 5×5 window, that gives us 25 equations per pixel, then we will have

$$\forall \text{ pixel } \mathbf{p} \text{ in the 5 by 5 window}, \nabla I(\mathbf{p}) \cdot \begin{bmatrix} u & v \end{bmatrix}^\top + \partial_t I(\mathbf{p}) = 0 \tag{5.6}$$

Now we can expand the for all quantifier into a matrix operation,

$$\begin{bmatrix} \partial_x I(\mathbf{p}_1) & \partial_y I(\mathbf{p}_1) \\ \vdots & \vdots \\ \partial_x I(\mathbf{p}_{25}) & \partial_y I(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \partial_t I(\mathbf{p}_1) \\ \vdots \\ \partial_t I(\mathbf{p}_{25}) \end{bmatrix} = 0 \tag{5.7}$$

and we rearrange it into a standard linear system form ($A\mathbf{d} = \mathbf{b}$),[5.3]

$$\underbrace{\begin{bmatrix} \partial_x I(\mathbf{p}_1) & \partial_y I(\mathbf{p}_1) \\ \vdots & \vdots \\ \partial_x I(\mathbf{p}_{25}) & \partial_y I(\mathbf{p}_{25}) \end{bmatrix}}_{\triangleq A} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{\triangleq \mathbf{d}} = - \underbrace{\begin{bmatrix} \partial_t I(\mathbf{p}_1) \\ \vdots \\ \partial_t I(\mathbf{p}_{25}) \end{bmatrix}}_{\triangleq \mathbf{b}} \tag{5.8}$$

5.3.1 Lucas-Kanade Equation: Least Squares Solution

Least Squares Solution for the motion vector $\mathbf{d}$ is given by

$$\begin{aligned} &A^\top A\mathbf{d} = A^\top \mathbf{b} \quad \Longleftrightarrow \\ &\begin{bmatrix} \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_x I(\mathbf{p}) & \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_y I(\mathbf{p}) \\ \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_y I(\mathbf{p}) & \sum_{\mathbf{p}'} \partial_y I(\mathbf{p}) \partial_y I(\mathbf{p}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_t I(\mathbf{p}) \\ \sum_{\mathbf{p}'} \partial_y I(\mathbf{p}) \partial_t I(\mathbf{p}) \end{bmatrix} \end{aligned} \tag{5.9}$$

where the summations are over all pixels in the $K \times K$ window. (In our case, 5×5). Equation 5.9 is called the Lucas-Kanade Equation.

[5.3] $A \in M_{25\times 2}(\mathbb{R}), \mathbf{d} \in M_{2\times 1}(\mathbb{R}), \mathbf{b} \in M_{25\times 1}(\mathbb{R})$. Here $\mathbf{d}$ is the motion vector that is unknown and we want to solve for it.

5.3.2 Connection: Second Moment Matrix

We can notice that

$$A^\top A = \begin{bmatrix} \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_x I(\mathbf{p}) & \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_y I(\mathbf{p}) \\ \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_y I(\mathbf{p}) & \sum_{\mathbf{p}'} \partial_y I(\mathbf{p}) \partial_y I(\mathbf{p}) \end{bmatrix} \tag{5.10}$$

is precisely the second moment matrix.

5.3.3 Solvability Condition

Optimal motion vector satisfies Lucas-Kanade Equation (Equation 5.9). It takes the form $A^\top A\mathbf{d} = A^\top \mathbf{b}$ and then ideally, we can solve $\mathbf{d} = (A^\top A)^{-1}(A^\top \mathbf{b})$. Clearly it is not always ideal, so we summarize the solvability conditions as follows:

- $A^\top A$ is invertible, i.e $\det(A^\top A) \neq 0$
- $A^\top A$ should not be too small due to noise: eigenvalues λ_1, λ_2 of $A^\top A$ should not be too small[5.4]
- $A^\top A$ should be well conditioned: $\max\{\lambda_1, \lambda_2\}/\min\{\lambda_1, \lambda_2\}$ should not be too large.

Importance of Eigenvectors The eigenvectors here summarizes the distribution of gradient of the image I, within this window of $K \times K$.

- Suppose that the two eigenvectors are such that $\lambda_1 \geq \lambda_2$, and they have respective eigenvectors $\mathbf{e}_1$ and $\mathbf{e}_2$. Then we know that $\mathbf{e}_1$ is the direction that is maximally aligned with the gradient of the image in the $K \times K$ window. Let's now further suppose that $\lambda_1 > 0 \wedge \lambda_2 = 0$, then the gradient is always a multiple of $\mathbf{e}_1$, since there is no eigen-compinent from $\mathbf{e}_2$ due to $\lambda_2 = 0$. This is the case if and only if I varies along with the direction $\mathbf{e}_1$ and is constant along $\mathbf{e}_2$. Edges!
- If $\lambda_1 = \lambda_2$, then the gradient in the $K \times K$ window has no prominent direction, which happens when we have rotational symmetry within the window!

5.3.4 Best Case and Pitfalls

Edges Causes Problems In real world images, slanted edges in images make things harder. Only one of the two eigenvalues will be small, and the other one will be large.

Low Texture Regions Don't Work When $\partial_x I$ and $\partial_y I$ are small (in terms of magnitude), we cannot easily recover motion. We will have small λ_1 and λ_2 in this case.

[5.4] RWe need this to be not close to singular, if it is close to singular, it's determinant will be close to zero, and the inverse will blowup.

High textured region works the best In this case gradients are different, with large magnitudes, and we have large λ_1 and λ_2.

5.3.5 Errors in Lukas-Kanade

What are the potential causes of errors in this procedure?

- Suppose $A^\top A$ is easily invertible
- Suppose there is not much noise in the image

When our assumptions are violated, we can easily catch them through our calculation,

- brightness constancy ***not*** satisfied
- the motion is ***not*** small
- A point does ***not*** move like its neighbours: window size is too large, but what is the ideal window size?

5.4 Iterative Refinement: Dealing w/ Larger Movements

The idea behind is to break down large movements into smaller ones, and we can let our Lucas Kanade solution handle the small movement. The procedure is as follows;

1. Initialize $(x', y') = (x, y)$
2. Compute (u, v) by

$$\begin{bmatrix} \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_x I(\mathbf{p}) & \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_y I(\mathbf{p}) \\ \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_y I(\mathbf{p}) & \sum_{\mathbf{p}'} \partial_y I(\mathbf{p}) \partial_y I(\mathbf{p}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_{\mathbf{p}'} \partial_x I(\mathbf{p}) \partial_t I(\mathbf{p}) \\ \sum_{\mathbf{p}'} \partial_y I(\mathbf{p}) \partial_t I(\mathbf{p}) \end{bmatrix} \quad (5.11)$$

   but with $\partial_t I = I(x', y', t+1) - I(x, y, t)$, where (x, y) is the original position.
3. Shift the window by (u, v): $x' = x' + u$; $y' = y' + v$
4. Recompute $\partial_t I$,
5. repeat steps 2 - 4 until small changes. Note: use interpolation for sub-pixel values.

I think this means small motion displacement.

$u(next) = u(current) + u(prev)$; $v(next) = v(current) + v(prev)$ and they tends to zero.

5.5 Coarse to Fine Optical Flow Estimation

How much motion is "small"? This is hard to answer, but we are sure if it is in terms of just a few pixels (maybe 1) then the motion is small. We can construct a gaussian image pyramid for each frame, and at a downsampled layer, we perform iterative Lucas Kanade. We then refine, using calculations, the motion detected to the original image.

A Few Details

- Top Level
  - Apply L-K to get a flow field representing the flow from the frist frame to the second frame,
  - apply this flow field to warp the first frame toward the second frame
  - rerun the L-K on the new warped image to get a flow field from it to the second frame.
  - repeat until convergence
- Next Level
  - Upsample the flow field to the next level as the first guess of the flow at that level
  - apply this flow field to warp the first frame towards the second frame
  - rerun L-K and warping till convergence as above
- etc

6 Scale Invariant Interest Point Detection

In last section, we talked about corner detection. If we zoom in on an area that was classified as a corner, at some point, it might cease to be a corner and become an edge. Hence, corner location is not scale invariant!

End Goal is to be able to match an object in different images where the object appears in different scale, rotation, viewpoints, etc. How?[6.1]

[6.1] How can we independently select interest points in each image, such that the detections are repeatable across different scales?

6.1 Characteristics of Good Features

6.1.1 Repeatability

Repeatability means that the same feature should be found in several images despite geometric and photometric transformations.

6.1.2 Saliency

Saliency means each feature we compute should be distinctive. We don't want each point in image A correspond to many interest points in another image B. If this is the case, it is really hard to find the correct correspondence.

6.1.3 Compactness and Efficiency

We want out feature vectors "represent" fewer elements than the actual image pixels, and at the same time we want these features to represent something meaningful.

6.1.4 Locality

We always want out features to be computed from a local and small area from the image. Hopefully, this helps us handle occlusions. E.g. a car with two wheels with a feature vector representing one of the wheels, and now if half of the car is block by something, the algorithm can still recognize this car model from the wheel it is able to see.

6.2 Blob Detection

Definition (Blob) A blob in an image is a patch of pixels that share some common property. e.g. they all have the same grayscale intensity values.

6.2.1 The Goal: Blob Filter

We wish to have a template of some sort, so that patches of the image could be compared against. We would like to have score that measure the similarity between a patch and template, and only be high when they match, and low otherwise. Scientist have drawn inspirations upon biological vision systems, particularly from a type of cell called centre-surround receptive field.[6.2]

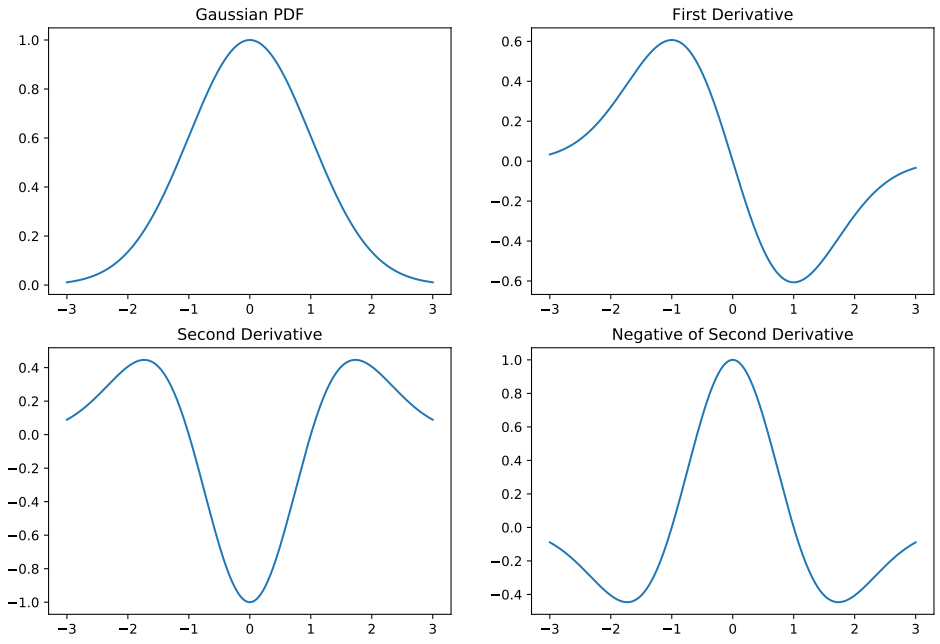

Figure 7: Construction of Mexican Hat Filter from Gaussian (1D Example)

6.2.2 Mexican-hat Shaped Filter 1-D Intuition

By taking second derivative of the Gaussian density, and taking negative we get the Mexican Hat filter (1D). Figure 7 illustrates the process. Around the centre of this hat, we will see positive values, e.g. approx. from -1 to 1 in our example. And if we go beyond the threshold, ($> 1 \vee < -1$) the the filter has negative values. This makes this function perfect for the role of blob detector! The 2D case can be easily obtained by revolving the 1D Mexican hat around zero point.

6.2.3 Laplacian of Gaussian Filter (2-D Formal)

Suppose $G : \mathbb{R}^2 \to \mathbb{R}$ specifies a Gaussian density, then

$$(\text{LoG}(x, y) =) \, L(x, y) \triangleq \Delta G = \partial_{xx} G + \partial_{yy} G \tag{6.1}$$

is the laplacian[6.3] of Gaussian. Now let's derive this. We start with the 2D gaussian density (assuming isotropic density, i.e. $\Sigma = \sigma I$), which is

$$G(x, y) = \underbrace{\frac{1}{2\pi\sigma^2}}_{c \in \mathbb{R}} \exp \underbrace{\left\{ -\frac{x^2 + y^2}{2\sigma^2} \right\}}_{f(x,y)} = c e^{f(x,y)} \tag{6.2}$$

Then,

$$\partial_x G = c \partial_x f e^{f(x,y)} \quad \implies \quad \partial_{xx} G = c \left(\partial_{xx} f e^{f(x,y)} + \partial_x f \partial_x f e^{f(x,y)} \right) \tag{6.3}$$

$$= c \left(-\frac{1}{\sigma^2} e^{f(x,y)} + \frac{x^2}{\sigma^4} e^{f(x,y)} \right) \tag{6.4}$$

$$= c \left(\frac{x^2 - \sigma^2}{\sigma^4} \right) e^{f(x,y)} \tag{6.5}$$

and similarly,

$$\partial_{yy} G = c \left(\frac{y^2 - \sigma^2}{\sigma^4} \right) e^{f(x,y)} \tag{6.6}$$

then arrive at our final form

$$L(x, y) = c \left(\frac{x^2 - \sigma^2}{\sigma^4} \right) e^{f(x,y)} + c \left(\frac{y^2 - \sigma^2}{\sigma^4} \right) e^{f(x,y)} \tag{6.7}$$

$$= \frac{1}{\pi\sigma^4} \left(\frac{x^2 + y^2}{2\sigma^2} - 1 \right) \exp \left\{ -\frac{(x^2 + y^2)}{2\sigma^2} \right\} \tag{6.8}$$

[6.2] with an excitatory centre and an inhibitory surrounding. They have been identified with usages such as edge enhancement, which enables human to perform better in detection and localization, and tracking of small objects.

[6.3] Note: Laplacian operator is a second order differential operator in the 2D Euclidian space. It is defined as the divergence of the gradient of the function. In our case, the function is gaussian density. $\Delta \mathbf{f} = \nabla^2 \mathbf{f} = \nabla \cdot \nabla \mathbf{f}$

The above final result will give us the inverse Mexican Hat ($\mathbb{R}^2 \to \mathbb{R}$).

6.2.4 LoG and Derivative of Gaussian wrt Variance

Recall that

$$\text{LoG}(x,y) = \frac{1}{\pi\sigma^4}\left(\frac{x^2+y^2}{2\sigma^2} - 1\right)\exp\left\{-\frac{(x^2+y^2)}{2\sigma^2}\right\} \tag{6.9}$$

and

$$G(x,y) = \underbrace{\frac{1}{2\pi\sigma^2}}_{c\in\mathbb{R}}\exp\underbrace{\left\{-\frac{x^2+y^2}{2\sigma^2}\right\}}_{f(x,y)} = ce^{f(x,y)} \tag{6.10}$$

clearly they look similar to one another. Now we want to find how they are connected mathematically. Staring at the equation, let's start with making them look them by cancelling and constants in front. We have

$$\text{LoG}(x,y) = G(x,y,\sigma)\frac{2\pi\sigma^2}{\pi\sigma^4}\left(\frac{x^2+y^2}{2\sigma^2} - 1\right) \tag{6.11}$$

$$= G(x,y)\frac{1}{\sigma^2}\left(\frac{x^2+y^2}{\sigma^2} - 2\right) \tag{6.12}$$

We can also derive that

add derivation to this, homework.

$$\frac{\partial G}{\partial\sigma} = \frac{1}{\sigma}\left(\frac{(x^2+y^2)}{\sigma^2} - 2\right)G(x,y,\sigma) \tag{6.13}$$

but wait! The above two equations look more than "just similar", in fact

$$\partial_\sigma G = \sigma\,\text{LoG}(x,y) = \sigma\Delta G(x,y) \tag{6.14}$$

6.2.5 Difference of Gaussians

By definition, the difference of gaussians is calculated as

$$\text{DoG}(x,y,\sigma,k) = G(x,y,\sigma) - G(x,y,k\sigma) \tag{6.15}$$

$$= \frac{1}{2\pi\sigma^2}\exp\left\{-\frac{x^2+y^2}{2\sigma^2}\right\} - \frac{1}{2\pi k^2\sigma^2}\exp\left\{-\frac{x^2+y^2}{2k^2\sigma^2}\right\} \tag{6.16}$$

Then,

$$\frac{-1}{k-1}\,\text{DoG} = \frac{G(x,y,\sigma) - G(x,y,k\sigma)}{\sigma - k\sigma}\sigma \tag{6.17}$$

$$\overset{k\to 1}{\approx} \frac{\partial G}{\partial\sigma}\cdot\sigma \tag{6.18}$$

Recall from Equation 6.14, then we know

$$\text{DoG} \approx (1-k)\sigma\frac{\partial G}{\partial\sigma} = (1-k)\sigma^2\Delta G(x,y). \tag{6.19}$$

6.2.6 Approximating Laplacian of Gaussian w/ Difference of Gaussians

To see how the above two sections relate, we will need one example. Suppose we have an image $I(x, y)$, $\sigma(s) = 2^s$, $R(x, y, \sigma) = (G(x, y, \sigma) * I(x, y))$. Now let's take the derivative of R with respect to the scale parameter s here,

$$\frac{\partial R}{\partial s} = \frac{\partial G}{\partial s} * I(x, y) \tag{6.20}$$

$$= \log 2\sigma(s) \left(\frac{\partial G}{\partial \sigma} * I(x, y) \right) \tag{6.21}$$

$$= \log 2\sigma^2 \Delta G(x, y) * I(x, y) \tag{6.22}$$

Now notice the final result from Section 6.2.5 Difference of Gaussians contains a similar filter as what we just derived. Scientist uses difference of gaussians as approximation for laplacian of gaussian filter.

But Why Isn't LoG Good Enough? The LoG filter is a non-separable one, and things get especially bad when we need a filter of large size. The laplacian of gaussian filter however, is a lot faster in terms of computation time since it is separable.

6.2.7 Applying Laplacian of Gaussian 1-D

When we try to use laplacian of gaussian to detect blob, we first convolve the input signal with the LoG with some arbitrary σ. Then we gradually increase the sigma, and find extremes that are scale invariant in terms of σ.

6.2.8 Characteristic Scale

We define the characteristic scale as the scale that produces peak (min / max) of the Laplacian response. This means that to detect a blob of certain size, we will need a Laplacian of Gaussian of a certain "scale" - the characteristic scale!

6.2.9 Applying Laplacian of Gaussian 2-D

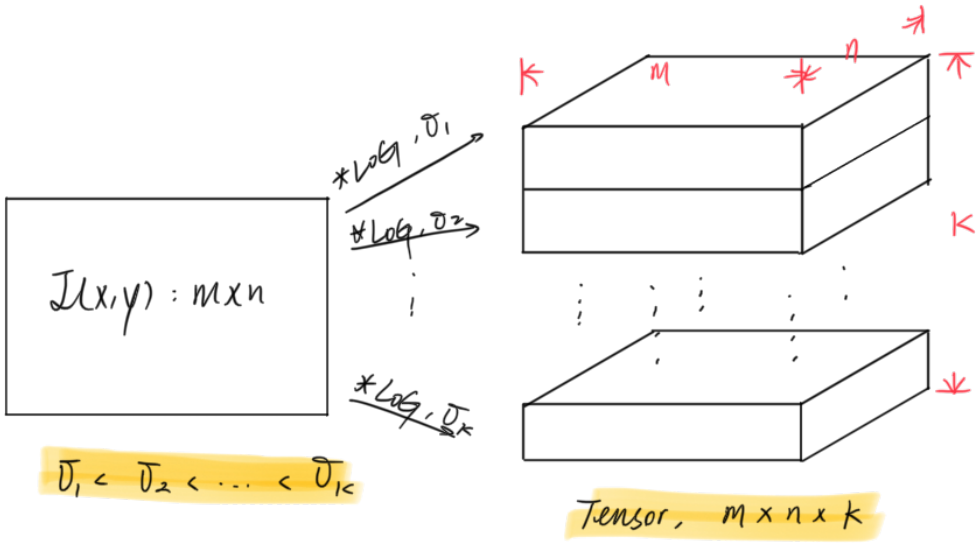

Figure 8: Using Laplacian of Gaussian to find interest points in 2D case (images). Interest points are local maxima in both position and scale.

Applying Laplacian of Gaussian in 2D image case is similar to the 1D case presented earlier, just this time we have an array of output maps (really a tensor). Figure 8 illustrates the input and output. The interest points that we want to find are local maxima in both position and scale, i.e. interest points are consistent along vertical direction in the output tensor.

6.2.10 Lowe's Difference of Gaussian

First compute a Gaussian image pyramid, and then compute the Difference of Gaussians

$$D(x, y, \rho) = I(x, y) * (G(x, y, k\rho) - G(x, y, \rho)) \tag{6.23}$$

for $\rho = \left\{\sigma, k\sigma, k^2\sigma, \ldots, k^{s-1}\sigma\right\}$ where $k = 2^{1/s}$. The application process is quite similar to the LoG one we discussed above, we will need to compute

$$I_s = I * G_{k^s\sigma} \tag{6.24}$$

$$\vdots \tag{6.25}$$

$$I_2 = I * G_{k^2\sigma} \tag{6.26}$$

$$I_1 = I * G_{k^1\sigma} \tag{6.27}$$

$$I_0 = I * G_{k^0\sigma} \tag{6.28}$$

Then we compute their differences (element-wise). There should be s difference maps, i.e. $I_1 - I_0, I_2 - I_1, ...$ Once we are done with the current scale, we will do it for the next one. (this is why we need the gaussian pyram id in the first place.) Finally, we find local maxima in scale, and do a bit of pruning of bad maxima.[6.4] We are done!

6.2.11 Other Interest Point Detectors

- Lindeberg: Laplacian of Gaussian
- Lowe: DoG (Typically called the SIFT interest point detector)
- Mikolajczyk & Schmid: Hessian / Harris laplacian / Affine
- Tuyttelaars & Van Gool: EBR and IBR
- Matas: MSER
- Kadir & Brady: Salient Regions

7 Scale Invariant Feature Transform (SIFT)

7.1 Step 1: Difference of Gaussians as Feature Detector

The first step is the same as described in section Lowe's Difference of Gaussian. First compute a Gaussian image pyramid, and then compute the Difference of Gaussians

$$D(x, y, \rho) = I(x, y) * (G(x, y, k\rho) - G(x, y, \rho)) \tag{7.1}$$

for $\rho = \left\{\sigma, k\sigma, k^2\sigma, \ldots, k^{s-1}\sigma\right\}$ where $k = 2^{1/s}$. The application process is quite similar to the LoG one we discussed above, we will need to compute

$$I_s = I * G_{k^s\sigma} \tag{7.2}$$

$$\vdots \tag{7.3}$$

$$I_2 = I * G_{k^2\sigma} \tag{7.4}$$

$$I_1 = I * G_{k^1\sigma} \tag{7.5}$$

$$I_0 = I * G_{k^0\sigma} \tag{7.6}$$

Then we compute their differences (element-wise). There should be s difference maps, i.e. $I_1 - I_0, I_2 - I_1, ...$

[6.4] Look over every x and y, what is the highest response in the scale space? We will keep all points with response greater than a certain threshold.

7.2 Step 2: Extracting Feature Points

For each key-point, we take the Gaussian-blurred image at corresponding scale ρ. Notice that this information needs to be stored as a pair. We will need this in the following steps.

7.3 Step 3: Gradient Magnitude and Orientation

Compute the gradient magnitude and the orientation in neighbourhood of each keypoint proportional to the detect scale. To achieve so, we first blur the image with gaussian wth respective ρ,

$$I * G_\rho \tag{7.7}$$

Then, compute, on a 16×16 neighbour around the key-point, the magnitude of gradients

$$|\nabla I(x,y)| = \sqrt{\left(\frac{\partial\left(I(x,y) * G_\rho\right)}{\partial x}\right)^2 + \left(\frac{\partial\left(I(x,y) * G_\rho\right)}{\partial y}\right)^2} \tag{7.8}$$

and the gradient orientation

$$\theta(x,y) = \arctan\left(\frac{\partial I * G_\rho}{\partial y} / \frac{\partial I * G_\rho}{\partial x}\right) \tag{7.9}$$

7.4 Step 4: Dominant Orientation Extraction

In this step, we wish to weigh gradients closer to the centre of the 16×16 patch higher. We can achieve so by element-wisely scale the gradient magnitudes according to a gaussian density

$$\nabla I(x,y) \mid \cdot G_{1.5\rho}(d) \tag{7.10}$$

then we compute a histogram of gradient orientations, each bin covers 10 deg. The orientation giving the peak in the histogram is the key point's orientation.

7.5 Step 5: Computing the Feature Vector

Compute a 128 dimensional descriptor: 4×4 grid, each cell is a histogram of 8 orientation bins relative to dominant orientation. Each descriptor has

$$P_i = (x_i, y_i, \rho_i, \vartheta_i) \quad \text{and} \quad f_i = ... \in \mathbb{R}^{128} \tag{7.11}$$

where x_i, y_i marks the location, ρ_i marks scale, ϑ_i is the orientation and f_i is the feature vector. This step is a bit abstract and hard to follow, let's dive deeper. Figure 9 illustrates the process. Each 4×4 component in the original 16×16 window contributes to the 128 dim[7.1] vector as one unit, via a 8 bin histogram.

[7.1] The original 16×16 grid is now treated as a 4×4, since each 4×4 subunit is now treated as one entity. There is one 8 bin histogram in each 4×4 grid, and thus we have $4 \times 4 \times 8 = 128$ bin values in total.

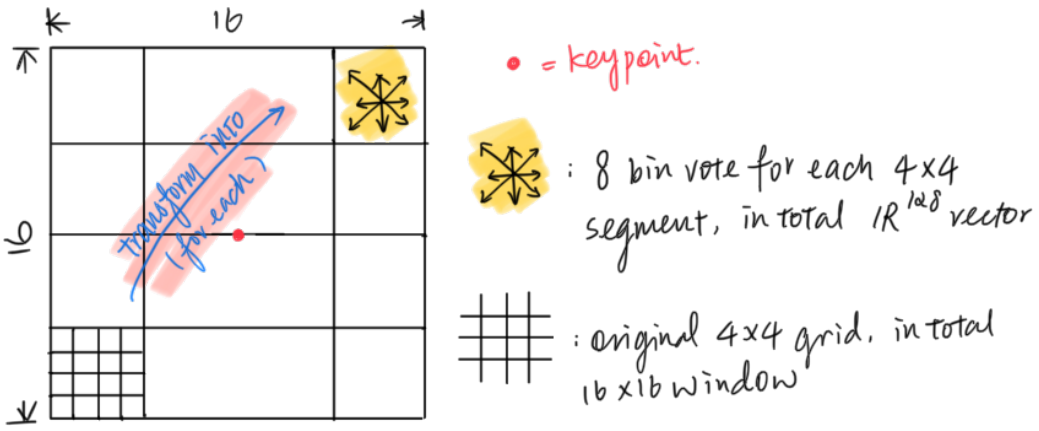

Figure 9: The 4×4 grid 8 bin vote transformation process.

7.5.1 Achieving Rotation Invariance

Our method as it is now is rotation dependent, i.e. if I rotate the original input image, everything will change. But how can we make it rotation invariant? At this point, we already have the dominant orientation from Step 4, so we just make the 8 bins based on the dominant orientation direction.

7.5.2 Achieving Illumination Independence

To achieve this, we just need to use some thresholding. Suppose we have an feature vector `f1` $\in \mathbb{R}^{128}$, then[7.2]

$$\texttt{f2[j] = f1[j] > thres ? thres : f1[j], for each j} \tag{7.12}$$

Finally, we need to normalize `f2`,

$$\texttt{f = (f2 .- mean(f2)) ./ normL2(f2)} \tag{7.13}$$

7.6 Properties of SIFT

SIFT is invariant to

- scale, and
- rotation

SIFT is partially invariant to

[7.2] I am using pseudo-code here just to make everything easier...

- Illumination changes (sometimes even day vs night)
- Camera viewpoint (up to about 60 deg of freedom of out of plane rotation.
- Occlusion, clutter

why?

Also it is important that SIFT is fast and efficient, can be run in real time! There are also a lot of code available :D.

7.7 PCA-SIFT

The dimensionality of SIFT is pretty high, we need 128 dimensions for each key point. We can try to reduce he dimensionality using linear dimensionality reduction such as Principle Component Analysis (PCA). We can reduce the dimensionality of each descriptor to 10 or so with PCA.

7.8 Matching Local Descriptors

So far, we have talked about SIFT, which computes a bunch of local descriptors for interest points. But now we want to match the features between pairs of images. Ideally, a match is a correspondence between a local part of the object on one image to the same local part of the object in another image. The naïve approach on this is to brute force compare all possible pairs of interest points detected in two images, and see if we have a match by computing a distance metric such as Euclidean distance. But can we rely solely on this min Euclidean distance criteria as the indicator of a match? There will be cases that several close matches exist, which essentially means that the system is not confident on one choice. To address this, we introduce the ratio

$$\phi_i = \frac{\|f_i - f_i'^*\|}{\|f_i - f_i'^{**}\|} = \frac{\text{Distance to closest}}{\text{Distance to second closest}} \tag{7.14}$$

where $f_i'^*$ is the closest and $f_i'^{**}$ is the second closest. If ϕ_i is small enough, then we have a reliable match. Otherwise if ϕ_i is large, then the system is unconfident. Notice that

- High threshold introduces false positives, i.e. incorrect matches being returned, while
- Low threshold results in false negatives, i.e. too many correct matches being missed.

Typically, we pick $\phi_i < 0.8$

8 Planar Objects Matching in New Viewpoints

8.1 Review: Linear Transformations

Linear transformations / linear maps are combinations of Scale, rotation, shear and mirror. A transformation T maps from one vector space into another one, and $T : V \to W$ is linear

iff $\forall \mathbf{u}, \mathbf{v} \in V, k \in \mathbb{R}$

- $T(\mathbf{u} + \mathbf{v}) = T(\mathbf{u}) + T(\mathbf{v}) \in W$
- $T(k\mathbf{u}) = kT(\mathbf{u})$

In our context of image, and in our task of finding the transformation between viewpoint changes of images, we will be dealing with 2×2 images, and the linear map will be achieved via a matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{8.1}$$

And there are several canonical transformation matrices (form) that get used often

Scaling Matrix

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{8.2}$$

Rotation Matrix Counter-clock wise rotation θ radian of the original vector space.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{8.3}$$

Shear Matrix 'distorts' rectangles into parallelograms.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left[\begin{array}{cc|c} 1 & \lambda & 0 \\ 0 & 1 & \end{array}\right] \begin{bmatrix} x \\ y \end{bmatrix} \tag{8.4}$$

Mirror Matrix Let

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \quad B = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{8.5}$$

then, A mirrors both x and y directions, and B, C mirrors only x or only y respectively.

8.1.1 Properties of Linear Transformations

- Origin always maps to origin, and
- lines always map to lines, and
- parallel lines remain parallel, and
- ratios are preserved, and
- closed under composition, meaning that we can collapse a serial of matrix together before applying it. Often, this is computationally more efficient.

8.2 Affine Transformations

So far, we've discussed linear transformations, which only accounts for transformation not translation. To address, we introduce the affine transformation, which takes the general form of

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{8.6}$$

8.2.1 Properties of Affine Transformations

- Origin does not necessarily map to origin.
- Lines map to lines, and
- parallel lines remain parallel
- ratios are preserved, and
- closed under composition, and
- rectangles go parallelograms

8.2.2 Approximating View Point Changes

Affine transformation approximates viewpoint changes for roughly planar objects and roughly orthographic cameras.

8.2.3 Computing the Affine Transformation

Let (x_i, y_i) be a point on the reference (model) image and x_i', y_i' its match in the test image. An affine transformation A maps (x_i, y_i) to (x_i', y_i'):

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{8.7}$$

where the matrix with elements $a, b, ..., f$ is a unknown matrix. This system gives us 2 equations, but we have 6 unknowns to solve for. To address, we can rewrite this into a simple linear system. (still 2 equations with 6 unknowns)

$$\begin{bmatrix} x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_i' \\ y_i' \end{bmatrix} \tag{8.8}$$

but the catch is that we have more than one match! and for each match we have two equations. We can stack all the equations up[8.1] and make it a bigger system, still with just 6 unknowns. Let's call this stacked system as

$$P\mathbf{a} = \mathbf{p}' \tag{8.9}$$

Notice that here it suffices to have three matching points to solve for the affine transformation. If we have three matches, then computing A is really easy:

$$\mathbf{a} = P^{-1}\mathbf{p}' \tag{8.10}$$

However, the more matching points data we have, the more reliable it will be, and in this case, we do the least squares estimation

$$\min_{a,b,\ldots,f} \left\| P\mathbf{a} - \mathbf{p}' \right\|_2^2 \tag{8.11}$$

which has a closed form solution

$$\mathbf{a} = (P^\top P)^{-1} P^\top \mathbf{p}' \tag{8.12}$$

where $(P^\top P)^{-1} P^\top$ is called Moore-Penrose Pseudo-inverse.

8.2.4 RANdom SAmple Consensus (RANSEC)

In previous sections, we assumed that there will be good correspondence between points. However, in reality, this might not be true, and we might suffer from outliers. In particular, least square estimation is very sensitive to outliers. RANSAC is here to the rescue. The approach is quite simple, and is as follows:

- Take the minimal number of points to compute what we want. In a Simple Linear Regression (SLR) line example, two point. And in our affine example of matching two images, three matches.
- By "take" we mean choose at random from all points
- fit a line to the selected pair of points
- count the number of all points that agree with[8.2] the line: calling them inliers.
- repeat this many times, remember the number of inliers for each trial, and the respective points selected for the generation of that specific line
- among several trials, select the one with the largest number of inliers.

[8.1] Vertically stack the matrices

[8.2] within some δ neighbourhood of the line

How many iterations?

- Suffience number of trials S must be tried
- Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials.
- The likelihood in one trial that all k random samples are inliers is p^k, then
- At least one one the k points is an outlier has probability $(1 - p^k)$,
- All S trials are bad has probability $(1 - p^k)^S$, then
- the probability of a least one is good is

$$P = 1 - (1 - p^k)^S \tag{8.13}$$

- and it is possible to solve

$$S = \frac{\log(1 - P)}{\log(1 - p^k)} \tag{8.14}$$

where $P \uparrow \implies S \uparrow$, and $p \downarrow \implies S \uparrow$, and $k \uparrow \implies S \uparrow$

9 Camera Models

Images are 2D projections of the real world scene, it captures two kinds of information

- Geometric: positions, points, lines, curves, etc
- Photometric: intensity, color

There is a complex 2D - 3D relationship between the 2D image captured by an camera and the actual scene which is in 3D. We use the camera model to approximate these relationships. We will discuss how are 3D primitives projected onto the image plane. We can do this using a linear 3D to 2D projection matrix.

9.1 Pinhole Camera Model

The distance from the camera pinhole to the image plane (which has the upside down, left-side right image projected) is called the focal length, denoted f. A virtual image plane is between the camera pinhole and the object in the world. It will be exactly focal length away from the camera centre, and it will have the exact same sized image as the image plane. (except this one is not upside down and left-side right.) It is easier to consider this plane. Now the goal is to find how the object in the world projects to the (virtual) image plane.

9.2 Camera / Image Coordinate System

We define the camera pinhole as the origin $\mathbf{0} \in \mathbb{R}^3$, it is called the optical centre. We use the right hand rule for specification of axis positive orientations. That is, from the perspective of the camera, towards left means positive in X direction, and upwards means positive in Y direction and further away from the camera (increase in depth of field) corresponds to positive direction in Z.

The Z axis is given the name of **Optical Axis** or **Principal Axis**. It is orthogonal to the (virtual) image plane between the object in the world and the camera centre. The camera coordinate axes X, Y are parallel to the image plane. The principal point $\mathbf{p}$ denotes the point where the principal axis intersects with the image plane. Note that $\|\mathbf{p}\|$ is equal to the focal length.

We denote the image axes with x and y. An image we see is of course represented with these axes. We call this an image coordinate system.

9.3 Projection

Suppose we have a point $Q \in \mathbb{R}^3 = (\mathbf{x}, \mathbf{y}, \mathbf{z})^\top$. (suppose that it $\mathbf{z} > f$) Then, using similar triangles (actually 3D similar triangles in this case), we know that this Q will appear on the virtual image plane at

$$\left(\frac{f \cdot \mathbf{x}}{\mathbf{z}}, \frac{f \cdot \mathbf{y}}{\mathbf{z}}, f \right)^\top \tag{9.1}$$

The notion above is relative to the principal point $\mathbf{p}$, but on a 2D projection the depth information is extraneous, and we can disregard it. We do so by essentially moving the origin to $(0, 0)$ in the image. Let $\mathbf{p}$ have coordinates (p_x, p_y) on the image plane in terms of the image coordinate system, then the projection is

$$\left(\frac{f \cdot \mathbf{x}}{\mathbf{z}} + p_x, \frac{f \cdot \mathbf{y}}{\mathbf{z}} + p_y, 0 \right)^\top \tag{9.2}$$

then the depth dimension is no longer useful and we have the actual projection onto the 2D image plane by throwing the last coordinate, i.e.

$$Q = (\mathbf{x}, \mathbf{y}, \mathbf{z})^\top \rightsquigarrow q = \left(\frac{f \cdot \mathbf{x}}{\mathbf{z}} + p_x, \frac{f \cdot \mathbf{y}}{\mathbf{z}} + p_y \right)^\top \tag{9.3}$$

Notice that this is not a linear transformation, since we have a division of $\mathbf{z}$ in in the new coordinate. (it is linear in terms of $1/\mathbf{z}$ but not in terms of $\mathbf{z}$). The problem now is how I can make this $\rightsquigarrow$ happen, but linearly.

We will use homogeneous coordinates (aka projective coordinates), which simply append 1 to the vector. For a 2D position (x, y), it becomes $(x, y, 1)$. Similarly for 3D positions, which will turn into a 4D vector.

Side Note: Homogeneous Coordinates A brief summarization of what homogeneous coordinates are ... borrowed from https://en.wikipedia.org/wiki/Homogeneous_coordinates#Notation

- Any point in the projective plane is represented by a triple (X, Y, Z), called the homogeneous coordinates or projective coordinates of the point, where X, Y and Z are not all 0.
- The point represented by a given set of homogeneous coordinates is unchanged if the coordinates are multiplied by a common factor.
- Conversely, two sets of homogeneous coordinates represent the same point if and only if one is obtained from the other by multiplying all the coordinates by the same non-zero constant.
- When Z is not 0 the point represented is the point (X/Z, Y/Z) in the Euclidean plane.
- When Z is 0 the point represented is a point at infinity.
- Note that the triple (0, 0, 0) is omitted and does not represent any point. The origin is represented by (0, 0, 1).

In homogeneous coordinates, scaling doesn't affect anything, i.e. $(x, y, 1)$ is congruent with (wx, wy, w) for $w \in \mathbb{R} \neq 0$. This fits the our projection line model, since all points along a projection line map to a single point $\mathbf{q}$ on the image. We can write what we want in Equation 9.3 homogeneous coordinates,

$$Q = (\mathbf{x}, \mathbf{y}, \mathbf{z})^\top \quad \rightarrow \quad q = \begin{bmatrix} \frac{f \cdot \mathbf{x}}{\mathbf{z}} + p_X \\ \frac{f \cdot \mathbf{y}}{\mathbf{z}} + p_y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f \cdot \mathbf{x} + \mathbf{z} \cdot p_x \\ f \cdot \mathbf{y} + \mathbf{z} \cdot p_y \\ \mathbf{z} \end{bmatrix} \tag{9.4}$$

and then we can write this as a matrix multiplication

$$Q = \begin{bmatrix} \mathbf{x}, \mathbf{y}, \mathbf{z} \end{bmatrix}^\top \rightarrow \begin{bmatrix} f\mathbf{x} + \mathbf{z}p_x \\ f\mathbf{y} + \mathbf{z}p_y \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} \tag{9.5}$$

where we denote

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{9.6}$$

is called the **camera calibration matrix** or **intrinsic parameter matrix**. The parameters in K are called **internal camera parameters**.

Finally,

$$K\left[\mathbf{x},\mathbf{y},\mathbf{z}\right]^\top = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} \to q = \begin{bmatrix} x \\ y \end{bmatrix} \implies \begin{cases} x = \frac{f\mathbf{x}}{\mathbf{z}} \cdot p_x \\ y = \frac{f\mathbf{y}}{\mathbf{z}} \cdot p_y \end{cases} \tag{9.7}$$

9.4 Camera Calibration Matrix

We presented that the Camera Calibration Matrix is denoted as K, and has values (this first formulation assumes square pixels)

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{9.8}$$

It can be a little more complicated, as pixels may not be square, (different focal length in x and y direction, denoted by f_x and f_y respectively.)

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{9.9}$$

and there might be a skew angle θ between x and y image axis

$$K = \begin{bmatrix} f_x & -f_x \cot\theta & p_x \\ 0 & f_y / \sin\theta & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{9.10}$$

9.5 Projection Properties

9.5.1 Many to One Projection

In the case of many to one projection, any points along same ray map to same point in image. Particularly,

- Points always map to points, and
- lines usually, though not always, map to lines. (A line going through the principle point maps to a point instead!)
- planes usually, though not always, map to planes. (A plane going through principal point projects to line)

wait shouldn't this be: principle axis maps to a point?

9.5.2 Vanishing Point

Parallel lines converge at a vanishing point. Each different direction in the world has its own vanishing point. All lines in the same direction in 3D intersect at the same vanishing point. Notice that this vanishing point is not necessarily inside the virtual image plane, but parallel lines in the world always converge to the same vanishing point.

A line that passes through V with direction D can be written as

$$X = V + tD \quad t \in \mathbb{R} \tag{9.11}$$

Then if we project it, we have

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = KX = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x + tD_x \\ V_y + tD_y \\ V_z + tD_z \end{bmatrix} = \begin{bmatrix} fV_x + ftD_x + p_xV_z + tp_xD_z \\ fV_y + ftD_y + p_yV_z + tp_yD_z \\ V_z + tD_z \end{bmatrix} \tag{9.12}$$

then we move infinitely far from the camera by taking $t \to \infty$ and compute x, y[9.1]

$$x = \lim_{t\to\infty} \frac{fV_x + ftD_x + p_xV_z + tp_xD_z}{V_z + tD_z} = \frac{fD_x + p_xD_z}{D_z} \tag{9.13}$$

$$y = \lim_{t\to\infty} \frac{fV_y + ftD_y + p_yV_z + tp_yD_z}{V_z + tD_z} = \frac{fD_y + p_yD_z}{D_z} \tag{9.14}$$

Importantly, this solution is independent of V, meaning that all lines with direction D go to this point.

***** Vanishing point can also be computed in another way. If we translate line with direction D to the camera centre, the the intersection of the line with the virtual image plane is the vanishing point corresponding to direction D!

We must also remember that vanishing points might not exist. In cases where lines parallel to image plane are also parallel in the image, they intersect at infinity! Also, for lines in the same 3D plane, the vanishing points lie on a line. We all it a vanishing line. The vanishing line for the ground plane is a horizon line.

9.6 Camera Extrinsics

The world is not described in camera coordinates, it is described by yet another system which is $(X_{world}, Y_{world}, Z_{world})$. The camera is placed at location c. Suppose our point of interest is Q, then clearly $Q - c$ makes the position relative to the camera. But the problem is what is Q in camera's coordinate system?

[9.1] when $t \to \infty$, fV_x, p_xV_z, V_z are all constants, and thus are small numbers. They don't matter. Same for the y ones.

Suppose u, v, w are three orthogonal directions of camera in room coordinate system. (much like a basis for movements of the camera.) Our goal is to have a rotation matrix R, that can take us from the world coordinate to the camera coordinate. Clearly the R is such that $R\begin{bmatrix} u & v & w \end{bmatrix} = I$ and $RR^\top = I$. Let

$$R = \begin{bmatrix} u^\top & v^\top & w^\top \end{bmatrix} \in M_{3\times 3}(\mathbb{R}) \tag{9.15}$$

Then, $\begin{bmatrix} X' & Y' & Z' \end{bmatrix}^\top$ in camera coordinates is

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = R\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} - c\right) = \begin{bmatrix} R & -Rc \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{9.16}$$

where $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}$ is in room / world coordinates.

9.7 Projection Equations

The projection matrix P models the culmulativeeffect of all intrinsic and extrinsic parameters. We use homogenous coordinates for 2D and 3D:

$$q = \begin{bmatrix} ax \\ ay \\ a \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{9.17}$$

It can be computed as

$$\mathrm{P} = \underbrace{\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsics } K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\underbrace{\begin{bmatrix} \mathrm{R}_{3\times 3} & 0_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} \mathrm{I}_{3\times 3} & \mathrm{T}_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}}_{\text{translation}}}_{\begin{bmatrix} R & -c \end{bmatrix}} \tag{9.18}$$

To get a point q in the image plane, I need to compute

$$P(X, Y, Z, 1)^\top \tag{9.19}$$

where $P \in M_{3\times 4}(\mathbb{R})$. This gives me a 3 by 1 vector. Now I divide all coordinates with the third coordinate (making the third coordinate equal to 1), and then drop the last coordinate.

The projection matrix can be compactly written as

$$P = K\begin{bmatrix} R & -c \end{bmatrix} \tag{9.20}$$

The (brutal) truth is in most cases you don't have P at all, so you can't really compute any projections. When you have a calibrated camera, then someone typically gives you P. And then projection is easy.