Total

Image Generation

Convolution mask should be reversed before using

Properties of convolution

- Commutativity $f * h = h * f$
- Associativity $f * (g * h) = (f * g) * h$
- Distributivity $f * (g + h) = (f * g) + (f * h)$
- Differentiation $\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx}$

Computational complexity

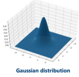
- Image size: $N \times N$
- Kernel size: $K \times K$
- What is the computational complexity?
  - At each pixel, K^2 multiplications, then K^2-1 summations
  - Do this for N^2 pixels
  - That is N^2K^2 multiplications and $N^2(K^2-1)$ summations
  - The complexity is $O(N^2K^2)$
- Can we accelerate this?

Computational complexity for separable filter

- Image size: $N \times N$
- Two kernels: first one $1 \times K$, second one $K \times 1$
- What is the computational complexity?
  - At each pixel, K multiplications, then $K-1$ summations
  - Do this for N^2 pixels
  - Do this twice for two kernels
  - That is $2N^2K$ multiplications and $2N^2(K-1)$ summations
  - The complexity is $O(N^2K)$, versus the original complexity $O(N^2K^2)$
- It makes a difference if K is large.

Gaussian Filter

- Kernel: 2D Gaussian distribution
$$h(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$
- The support is infinite, but we may ignore small values outside $[-k\sigma, k\sigma]$, e.g. $k = 3$.
- It is also a separable filter
$$h(i,j) = h(i) * h(j)$$
$$h(i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2}{2\sigma^2}}$$

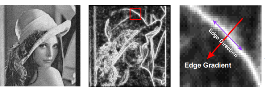

Gaussian distribution

0.003	0.013	0.022	0.013	0.003
0.013	0.060	0.098	0.060	0.013
0.022	0.098	0.162	0.098	0.022
0.013	0.060	0.098	0.060	0.013
0.003	0.013	0.022	0.013	0.003

Normalized coefficients of a 5x5 Gaussian kernel with σ=1.

Edge Detection

Edge Direction and Gradient



Gradient Magnitude $g(x) = \sqrt{g_x^2 + g_y^2}$

Gradient Direction $\tan^{-1}(g_y/g_x)$

$$h_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, h_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

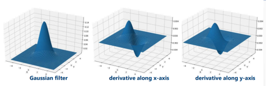
Edge detection


Magnitude map

- Given the filter magnitude maps, where exactly are the edges? Can we generate a binary edge map?
- Canny proposed the following criteria for an edge detector:
  - Good detection - There should be a low probability of failing to mark real edge points, and low probability of falsely marking non-edge points.
  - Good localization - The points marked as edge points by the operator should be as close as possible to the center of the true edge.
  - Single response - Only one response to a single edge.
- Canny proposed an edge detection algorithm, which can be broken down to the following steps:
  1. Perform Gaussian filtering to suppress noise;
  2. Calculate the gradient magnitude and direction;
  3. Apply non-maximum suppression (NMS) to get a single response for each edge;
  4. Perform hysteresis thresholding to find potential edges.

Gaussian filtering

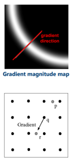
- Gaussian filtering is performed to smooth image and suppress noise.
- The gradient is calculated after Gaussian filtering.



Differentiation property of convolution:
$$\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx}$$

Non-maximum suppression (NMS)

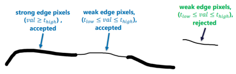
- A single response for an edge.
- Edge occurs where the gradient magnitude reaches the maximum.
- Check whether the pixel q is a local maximum along gradient direction.
- It may require checking interpolated pixels p and r.
- Alternatively, we can round the gradient directions into 8 possible angles.
  - 0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°
  - Horizontal, vertical, diagonal and anti-diagonal
- Suppression
$$M(x,y) = \begin{cases} M(x,y), & \text{local maximum} \\ 0, & \text{otherwise} \end{cases}$$



Gradient magnitude map

Hysteresis thresholding

- Thresholding
  - The most common method to convert an intensity image into a binary image.
$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) \geq t \\ 0, & \text{otherwise} \end{cases}$$
- Hysteresis thresholding
  - Define two thresholds t_{low} and t_{high} for edge detection
  - If a pixel gradient magnitude is larger than t_{high}, it is accepted as an edge.
  - If a pixel gradient magnitude is lower than t_{low}, it is rejected.
  - If in between, it may be an edge.
    - Consider its neighbouring pixels. It will be accepted if it is connected to an edge.



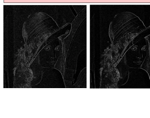
Effect of Gaussian kernel parameter

- The choice of σ depends on desired behavior
  - A large σ detects large scale edges.
  - A small σ detects fine features.

Second Order Detector (Laplace)

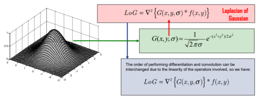
- The Laplace operator is usually used for approximating the second derivative of the image. A 3x3 mask is often used; for 4-neighbourhoods and 8-neighbourhoods it is defined as follows:

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad h = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

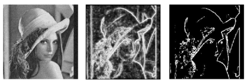


Marr-Hildreth Edge Detector

- From neurophysiological experiments, David Marr concluded in the seventies that object boundaries are the most important cues that link an intensity image with its interpretation. He proposed the use of zero crossings of the second derivative for accurate edge detection.
- The first derivative of the image function should have an extremum at the position corresponding to the edge in the image, so the second derivative should be zero at the same position, however, it is much easier and more precise to find a zero crossing position than an extremum.



Edge Based Segmentation



- Edge detection only provides a pixel-based representation of the likelihood of a pixel being an edge or not, it is not a geometrical representation of the boundary of the object;
- The simplest, and also the least effective method of grouping edges is to use heuristic search. That is to say, the algorithm starts from a single boundary pixel, and then attempts to join neighbouring pixels on the basis of their edge strength and direction.

As long as the gradient direction are within the threshold, we are grouping them

How to Group Edges?

- For a given image, let the gradient magnitude be specified by g and the gradient direction by angle φ. One can define heuristics based on gradient to indicate when two adjacent pixels can be joined if

$$|\varphi(x_i, y_i) - \varphi(x_j, y_j)| < \theta$$

You will need to make sure φ is within (-π, π), as well as the difference to avoid phase wrap-around (why?)

A predefined threshold value

Fourier Method

Fourier Methods – Mathematical Description

- A periodic function f(x+N)=f(x) can be approximated by an infinite sum of sinusoidal functions (cosine and sine), each with a frequency that is an integer multiple of 1/*N*.

$$f(x) = a_0 + a_1 \cos\theta + b_1 \sin\theta + a_2 \cos 2\theta + b_2 \sin 2\theta + \ldots$$

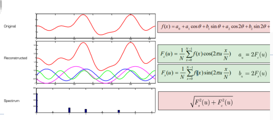
where θ=2πx/N, and 2π/N is called the fundamental frequency

$$f(x) = \sum_{h=0}^{\infty} a_h \cos(2\pi h \frac{x}{N}) + b_h \sin(2\pi h \frac{x}{N})$$

The purpose of Fourier transform is to find an easy way to recover the values of a_0, a_1, b_1, a_2, b_2, … To do this, we can make use of the orthogonal properties of trigonometric functions.

Sine components Fs(u) and Cosine components Fc(u)

Fourier Methods - Summary



$$f(x) = a_0 + a_1 \cos\theta + b_1 \sin\theta + a_2 \cos 2\theta + b_2 \sin 2\theta + \ldots$$

$$F_c(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\cos(2\pi u \frac{x}{N}) \quad a_u = 2F_c(u)$$

$$F_s(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\sin(2\pi u \frac{x}{N}) \quad b_u = 2F_s(u)$$

$$\sqrt{F_s^2(u) + F_c^2(u)}$$

Fourier Series Forms

Sine-cosine form
$$f(x) = \sum_{h=0}^{\infty} a_h \cos(2\pi h \frac{x}{N}) + b_h \sin(2\pi h \frac{x}{N})$$

Amplitude-phase form
$$f(x) = \frac{A_0}{2} + \sum_{h=1}^{N} \left(A_h \cos(2\pi h \frac{x}{N} - \varphi_h)\right)$$

Exponential form
$$f(x) = \sum_{h=-N}^{N} c_h e^{i2\pi h \frac{x}{N}}$$

$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$
$$\cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

Fourier Transform

- The Fourier Transform is a tool that converts a function into an alternative representation.
- The term *Fourier transform* refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of time.
- The Fourier transform of a function is a complex-valued function of frequency, whose magnitude represents the amount of that frequency present in the original function, and whose argument is the phase offset of the basic sinusoid in that frequency.
- Linear operations performed in one domain (time or frequency) have corresponding operations in the other domain, which are sometimes easier to perform. For example, convolution in the time domain corresponds to multiplication in the frequency domain. This means that operations can be performed in the frequency domain and the result can be transformed to the time domain.

Fourier Transform – in complex form

- The Discrete Fourier Transform is defined as:

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\left(exp\left(-2\pi ju\frac{x}{N}\right)\right)$$

Amplitude $|F(u)| = \sqrt{F_s^2(u) + F_c^2(u)}$

Phase $\varphi(u) = \tan^{-1}\frac{F_s(u)}{F_c(u)}$

- The inverse transform can be represented as:

$$f(x) = \sum_{u=0}^{N-1} F(u)\left(exp\left(2\pi ju\frac{x}{N}\right)\right)$$

Convolution operation in spatial domain becomes product operation in frequency domain

Fourier Transform – Basic Properties

	Spatial Domain (x)	Frequency Domain (u)
Linearity	$c_1 f(x) + c_2 g(x)$	$c_1 F(u) + c_2 G(u)$
Scaling	$f(ax)$	$\frac{1}{\lvert a\rvert} F\left(\frac{u}{a}\right)$
Shifting	$f(x - x_0)$	$e^{-i2\pi u x_0} F(u)$
Symmetry	$F(x)$	$f(-u)$
Conjugation	$f^*(x)$	$F^*(-u)$
Convolution	$f(x) * g(x)$	$F(u)G(u)$
Differentiation	$\frac{d^n f(x)}{dx^n}$	$(i2\pi u)^n F(u)$

Fourier Transform – Basic Properties

Spatial Domain		Frequency Domain
$g = f * h$	$\longleftrightarrow$	$G = FH$
$g = fh$	$\longleftrightarrow$	$G = F * H$

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

Rotation will only change phase, but not amplitude

Fourier Transform – Basic Properties

- The advantage of using the Fourier transform is in its invariant properties
- Rotating the object causes a phase change to occur, and the same phase change is caused to all the components

$$f(x + a) \Rightarrow F(u)e^{2\pi jau}$$

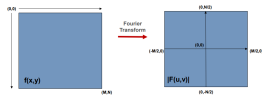
Fourier Transform – Basic Properties

- Scaling the object changes the magnitude of all the components by the same factor.
- If the magnitude of the spectrum is normalised, such that its maximum is equal to 1, then this normalised spectrum is invariant to object size.

$$af(x) \Rightarrow aF(u)$$

2D Fourier Transforms

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp(-2\pi ju\frac{x}{M})\exp(-2\pi jv\frac{y}{N})$$

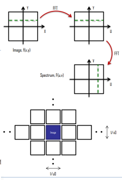


2. Purpose of 2D Fourier Transform

The 2D Fourier Transform decomposes an image into its **frequency components**, showing how the image can be reconstructed using sinusoidal patterns at different frequencies.

- **Low frequencies** ($u, v \approx 0$): Represent smooth, large-scale patterns or regions of uniform intensity.
- **High frequencies** (u, v **large**): Represent sharp changes or fine details, such as edges or textures.

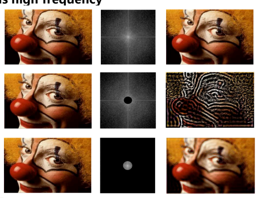
Separability of the 2D Fourier Transform



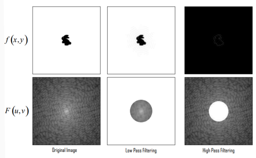
Again, a fast algorithm (Fast Fourier Transform - FFT) is available for computing this transform, providing that N and M are powers of 2. In fact, a two dimensional transform can be separated into a series of one dimensional transforms. In other words, we transform each horizontal line of the image individually to yield an intermediate form in which the horizontal axis is frequency u and the vertical axis is space y. We then transform individually each vertical line of this intermediate image to obtain each vertical line of the transformed image. Hence a two dimensional transform with an n by n image consists of 2n one dimensional transforms.

- In Fourier space, where is more of the information that we see in the visual world?
  - Amplitude
  - Phase
- The frequency amplitude of natural images are quite similar
  - Heavy in low frequencies, falling off in high frequencies
- Most information in the image is carried in the phase, not the amplitude
  - Not quite clear why

In the middle is low frequency, on the edge is high frequency



2D Fourier Transforms – A Pictorial Overview



Representation and Matching Cross correlation is comparing a template throughout the picture and check which area has the highest matching, before correlation, we should do preprocessing make sure both image and template has zero mean (shown in third equation)

Template Matching

- Cross-correlation defines a way of combining two functions. In the most general form it is defined as a continuous integral which in two dimensions is:

$$C(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(p,q)h(p+x, q+y)dpdq$$

The function h in the above equations is usually called a template

- For our purposes we are interested in discrete functions of finite size. For these, the cross-correlation integral simplifies to:

$$C(x,y) = \sum_{p=1}^{x_{res}}\sum_{q=1}^{y_{res}} f(p,q)h(p+x, q+y)$$

where f(x,y) is the discrete function defining the pixel values, h(u,v) is the template function, and [1..x_{res},1..y_{res}] is the range over which f(x,y) is non zero, i.e., the resolution of the picture. The normal restriction on the above equations is that the functions are symmetric about zero, i.e.,

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(p,q)dpdq = 0 \text{ and } \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(p,q)dpdq = 0$$

For any image, this may be arranged by subtracting the average value from each image point.

Cross correlation and convolution: In cross correlation, the kernel is unflipped, in convolution, the kernel is flipped.

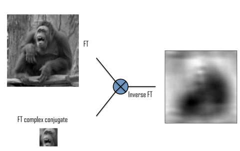
Convolution and Cross-Correlation

- If h is symmetric, i.e. $h[i,j] = h[-i,-j]$, convolution is equivalent to cross-correlation.

$$f[m,n] * h[m,n] = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f[m+i, n+j]h[-i,-j] = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f[m+i, n+j]h[i,j]$$

For Template matching the complex conjugate of the FT is used, which is different from convolution

Fourier Transform for Template Matching



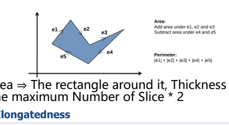
Shape Discriminants: Shape discriminants are simple object property that can be computed from the image and are independent of the object position Area: Counting pixel vs Area under edge vector

Area and Perimeter

- Object area and perimeter are the simplest features that can be extracted from the pixel image

Providing that the image is bi level, and that a single object is present in the image, we can of course compute the area by **summing all the pixels** of the camera image. This method is, however, inefficient since it requires every pixel in the image to be processed. Early systems **traced round the boundary** at the pixel level, but this was rather inaccurate, being prone to alias effects, particularly when trying to determine perimeters.

- In cases where a **piece wise boundary** has been determined using say a **Hough transform**, the area can be measured by processing the **edge vectors** of a polygon in order, adding the area under the edge vectors when moving to the right, and subtracting it when moving to the left. In this case, the perimeter can be computed accurately from the coordinates of the vertices.

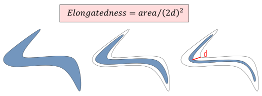


Area ⇒ The rectangle around it, Thickness ⇒ The maximum Number of Slice * 2

Elongatedness

- Elongateness can be evaluated as a ratio of the region area and the square of its thickness.
- The maximum region thickness can be determined as the number of mathematical **erosion steps** that may be applied before the region totally disappears.
- Elongatedness is independent of linear transformations - translation, rotation, and scaling.

$$Elongatedness = area/(2d)^2$$

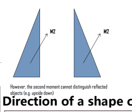


Moments m00 ⇒ Total number of pixels, the area m10 and m01 ⇒ Used to calculate the center of the rectangle m20 and m02 ⇒ Measure spread on an axis

1. μ_{20}:
   - Measures the spread along the **x-axis**.
   - A large μ_{20} means the object is elongated horizontally.
2. μ_{02}:
   - Measures the spread along the **y-axis**.
   - A large μ_{02} means the object is elongated vertically.
3. μ_{11}:
   - Measures the **correlation** between x- and y-coordinates.
   - Indicates the **tilt** or **orientation** of the object.

Second Moments

The second moment gives us a measure of how spread out the object is. If we consider the vector formed by the two components μ_2=[μ_{20}, μ_{02}], its magnitude is position independent, and so can be used as a discriminant, and its direction can be used to measure the orientation of the object. This is a useful property for control of a robot gripper. Note though, that the second moment is always positive, and hence the second moment cannot detect reflected objects (see below). However, the same fact means that μ_{20}+ μ_{02} may be used in place of the normal Euclidian distance for a magnitude discriminant. Rarely however do higher order moments yield any more discriminant power than the second.



However, the second moment cannot distinguish reflected objects (e.g. upside down)

$$Cov[f(x,y)] = \begin{bmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{bmatrix}$$

The eigenvectors of the covariance matrix correspond to the major and minor axes of the object.

Direction of a shape can be calculated by

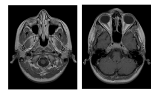
$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right)$$

How pk is calculated

- The probability of a grey level k is:
$$P_k = \frac{\text{Number of pixels with grey level } k}{\text{Total number of pixels in the image}}$$
- P_k is a normalized value such that:
$$\sum_{k=0}^{255} P_k = 1$$

Energy and Entropy

- Energy and entropy can be applied to objects where the shade or texture may have given characteristics.
- The image intensities need to be normalised before the discriminants are computed to account for varying lighting conditions

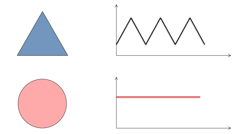


$$Energy = \sum_K P_k^2$$
$$Entropy = -\sum_K P_k \log(P_k)$$

Probability of grey level k

Signature

- The signature represents a shape using a one dimensional function derived from the shape's boundary points
- Many signatures exist; the ones shown below are the centroid distance



Interesting Point Detection Desired property of Features

- Locality – local features are associated with object features
- Generalisability – local features are more generalisable for scene representation
- Abundance – many of them, often with redundancy (numerically attractive)
- Efficiency – fast to calculate, thus good for real-time performance

- Well defined position in images;
- Incorporate useful local image context (e.g., distinctive texture patch);
- Stable under local or global changes (e.g., illumination, contrast);
- Immune to geometrical changes (e.g., scale, translation, rotation);
- Generalisable to natural scenes and preferably with good mathematical representation.

Edges	Corners	Blobs	Ridges
Sobel	Harris	LoG	Ridge sets
Prewitt	Shi & Tomasi	DoG	Valley sets
Roberts	Yang & Chabat	DoH (Determinant of Hessian)	Relative critical sets
Robinson	Level Curvature	MSER (Maximally stable extremal regions)	
Canny	Susan	PCBR (Principal curvature based region)	
Canny-Deriche	AST/FAST		

Why use corners: For uniform regions, any changes will result in minimum changes. For edges, moving along the edge will result in minimum changes. For corners, any movement will result in significant changes. Harris ⇒ Detecting changes on the x and y axes

Harris Corner Detector

- Let's look at a small window W, when shifted by **d**=[u, v]^T
- The changes in intensity distribution due to **d** can be expressed as the squared difference of I(*x*+*u*, *y*+*v*)-I(*x*,*y*) within the window

$$\varepsilon(u,v) = \sum_{x,y \in W}\left(I(x+u, y+v) - I(x,y)\right)^2$$

d=[u, v]^T

For uniform patches, this will be near 0 and for distinctive patches such as corners, it will be large. therefore we are interested in maximising ε(u,v) while searching for corners

By Taylor series expansion $I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \ldots$

$$\varepsilon(u,v) \approx \sum_{x,y \in W}\left(I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v - I(x,y)\right)^2, \text{ define } I_x \equiv \frac{\partial I}{\partial x} \text{ and } I_y \equiv \frac{\partial I}{\partial y} \text{ we have}$$

$$\varepsilon(u,v) \approx \sum_{x,y \in W}\left(uI_x + vI_y\right)^2 = \sum_{x,y \in W}\left(u^2I_x^2 + 2uvI_xI_y + v^2I_y^2\right)$$

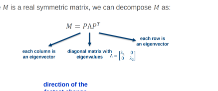
Direction of Harris

Harris Corner Detector

- Since M is a real symmetric matrix, we can decompose M as:

$$M = P\Lambda P^T$$

each column is an eigenvector; diagonal matrix with eigenvalues $\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$; each row is an eigenvector



The minimum and maximum eigenvalues of the Hessian can be interpreted as the (scaled) inverse variances in the least-certain and most-certain directions of motion.

Two eigenvectors will always be perpendicular.

Algorithm Design – Harris Corner Detection

- Compute x and y derivatives of image (can convolve image with derivatives of Gaussians);
- Compute products of derivatives at every pixel;
- Compute the sums of the products of derivatives;
- Construct the M matrix;
- Calculate the corner response value R;
- Threshold on value R, suppress non-maximum value.

$$I_x = G_x(x,y,\sigma) * f(x,y)$$
$$I_y = G_y(x,y,\sigma) * f(x,y)$$

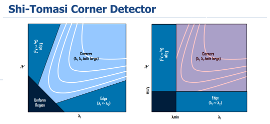
$$M = \begin{bmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2 \end{bmatrix}$$

Empirically determined constant

$$R = \det\mathbf{M} - k(trace\mathbf{M})^2$$

$$\det\mathbf{M} = \lambda_1\lambda_2 \qquad trace\mathbf{M} = \lambda_1 + \lambda_2$$

Shi-Tomasi Corner Detector



Empirically determined constant

$$R = \det\mathbf{M} - k(trace\mathbf{M})^2$$

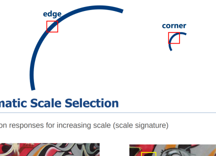
$$\det\mathbf{M} = \lambda_1\lambda_2 \qquad trace\mathbf{M} = \lambda_1 + \lambda_2$$

$$R = \min(\lambda_1, \lambda_2)$$

When the corner is rotated, the eigenvalues will not change, but the eigenvectors will change.

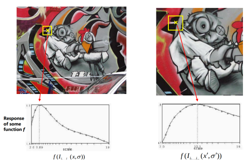
Corner Detector Invariance Properties

- When the image scale changes, the corner response will change.
- However, we can apply a corner detector at multiple scales and find the response at the most suitable scale.
- How do we normally get multi-scale images?
  - Gaussian smoothing with different σ
  - Sampling at different pixel resolutions



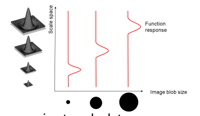
Automatic Scale Selection

- Function responses for increasing scale (scale signature)



Automatic Scale Selection

- The Laplacian of Gaussian (LoG) which is the second derivative of Gaussian is a suitable function to estimate the characteristic scale of image structures such as blobs and corners.
- When the size of the LoG kernel matches the size of an image structure the response attains an extremum.

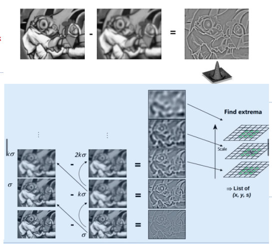


LOG is expensive to calculate; we can use DoG to approximate it.

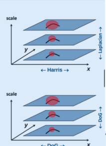
Approximate LoG with Difference-of-Gaussian (DoG).

$$LoG(x,y,\sigma) = \nabla^2 G(x,y,\sigma) = I_{xx}(x,y,\sigma) + I_{yy}(x,y,\sigma)$$
$$DoG(x,y,\sigma) = I * G(k\sigma) - I * G(\sigma)$$

1. Blur image with σ Gaussian kernel
2. Blur image with *k* σ Gaussian kernel
3. Subtract 2. from 1.



- Harris-Laplace detector [1]
  - Find the optimal scale σ using Laplacian response
  - Find the local maximum in space using Harris detector response
- SIFT [2]
  - Find the optimal scale σ using DoG response
  - Find the local extremum in space using DoG response



Feature Descriptors and Matching Matching Feature in different images

Image Processing with Local Feature

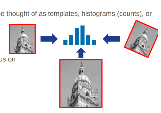
1. **Detection** - Find a set of distinctive key points.
2. **Description** - Extract feature descriptor around each interest point as vector.
$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$
$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$
3. **Matching** - Compute distance between feature vectors to find correspondence.
$$d(\mathbf{x}_1, \mathbf{x}_2) < T$$

Desired property of Feature Descriptor

Feature Descriptors

Note for good image features, we require **locality** (small areas are less sensitive to view-dependent deformations), **invariance** (scale, orientation and deformation), **repeatability** (same points can be repetitively identified as required for image sequence tracking), and **distinctiveness** (to ensure high sensitivity and specificity).

- Most feature descriptors can be thought of as templates, histograms (counts), or combinations
- The ideal descriptor should be
  - Robust and Distinctive
  - Compact and Efficient
- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color rarely used



SIFT Descriptor

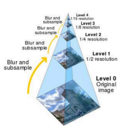
SIFT – Scale Invariant Feature Transform

Step 1: Scale-space Extrema Detection - Detect interesting points using DOG.

Step 2: Keypoint Localization – Determine location and scale at each candidate location, and select them based on stability.

Step 3: Orientation Estimation – Use local image gradients to assign orientation to each localized keypoint. Preserve theta, scale and location for each feature.

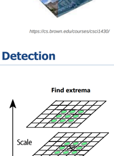
Step 4: Keypoint Descriptor - Extract local image gradients at the selected scale around the keypoint and form a representation invariant to local shape distortion and illumination



https://cs.brown.edu/courses/csci1430/

SIFT Descriptor - Extrema Detection

- Increasing σ increases robustness, but it is also computationally expensive. σ = 1.6 a good tradeoff
- It has been shown experimentally that the highest repeatability is obtained when sampling 3 scales per octave
- Detect maxima and minima of Difference-of-Gaussian in scale space
- Each point is compared to its 26 neighbors in the current image and in the scales above and below



SIFT Descriptor - Keypoint Localization

- For each candidate keypoint, interpolation of nearby data is used to accurately determine its position and scale
- This can be done by fitting a quadratic Taylor expansion of the Difference-of-Gaussian scale-space function shifted so that the origin is at the sample point

$$D(\mathbf{X}) = D + \frac{\partial D^T}{\partial \mathbf{X}}\mathbf{X} + \frac{1}{2}\mathbf{X}^T\frac{\partial^2 D}{\partial \mathbf{X}^2}\mathbf{X}$$

where $\mathbf{X} = (x, y, \sigma)^T$

- The offset for the extremum location is determined by taking the derivative of this function with respect to X and setting it to zero

$$0 = \frac{\partial D}{\partial X} + \frac{\partial^2 D}{\partial X^2}\hat{X}$$

SIFT Descriptor - Keypoint Localization

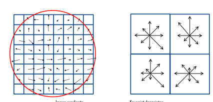
- Some of the detected keypoints lie along an edge, or they don't have enough contrast. In both cases, they are not as useful as features and should be eliminated
- If the DoG value at an extremum is less than a threshold (e.g 0.03), the keypoint is rejected

$|D(\hat{\mathbf{x}})| < 0.03$

https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

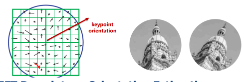
SIFT Descriptor

- Full Version
  - Divide the 16×16 window into a 4×4 grid of cells (8×8 window and 2×2 grid shown below for simplicity)
  - Compute an orientation histogram for each cell
  - 16 cells * 8 orientations = 128 dimensional descriptor



SIFT Descriptor - Invariance Properties

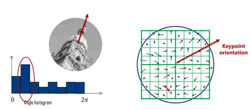
- To be robust to intensity value changes
  - Use gradient orientations
- To be scale-invariant
  - Estimate the scale using scale-space extrema detection
  - Scale the window size based on that scale at which the point was found
  - Calculate the gradient after Gaussian smoothing with this scale
- To be orientation-invariant
  - Rotate the gradient orientations using the dominant orientation in a neighbourhood.



SIFT Descriptor - Orientation Estimation

- To be orientation-invariant the descriptor of a keypoint can be computed relative to the dominant orientation in its neighbourhood

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = a\tan 2((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$



Properties of SIFT Descriptors for Matching

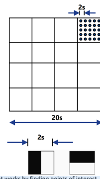
- Robust and can handle changes in viewpoint
  - Up to about 60 degrees out of plane rotation
- Can handle significant changes in illumination
- Fast and efficient, can run in real-time
- Other variations available, e.g., SURF, PCA SIFT, GLOH (gradient location-orientation histogram)

SURF: Speeded Up Robust Features

- To estimate the SURF descriptor of a keypoint:
  - Divide an image window of size 20*scale around the keypoint into 4x4 subregions
  - The Haar wavelet response d_x in the horizontal and d_y in the vertical direction is estimated in each subregion at 5x5 regularly spaced sample points
  - In each subregion:
    - estimate dx and dy at the 25 (5x5) points
    - sum over all 25 points to get 4 values

SURF = (dx, dy, |dx|, |dy|)

  - This results in a descriptor vector of length 4x16=64
  - 3 times faster than SIFT



- SURF is a feature detection and description algorithm that works by finding points of interest in an image and creating descriptors for them.
- It is designed to be:
  - **Scale-invariant**: It detects features regardless of their size in the image.
  - **Rotation-invariant**: It detects features regardless of their orientation.
  - **Fast and efficient**: It uses integral images and simplified calculations for speed.

Feature Descriptor Distance

Two feature descriptors **x** an **y** can be compared using any of the following distances:

Quadratic distance
$$D(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^2\right)$$

x^2 distance
$$D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i}$$

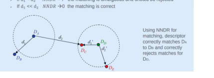
Cosine similarity
$$D(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{x_1y_1 + \cdots + x_ny_n}{\sqrt{\sum_{i=1}^n x_i^2}\sqrt{\sum_{i=1}^n y_i^2}}$$

$\theta = \arccos(x \cdot y / |x||y|)$

Feature Matching

1. Directly compare distance with a threshold
2. Match the nearest keypoint in the feature space
3. Using the nearest Neighbour Distance Ratio, compare the distance between the nearest neighbour and second nearest neighbour. If the ratio is around 1, reject; if it is approximately 0, then accept.

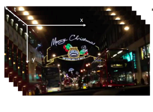
- Another feature matching approach is the **Nearest Neighbour Distance Ratio**
- Estimate the distance of a feature vector to its Nearest Neighbour (d_1) and its second Nearest Neighbour (d_2)
$$\text{NNDR} = \frac{d_1}{d_2} = \frac{|D_A - D_B|}{|D_A - D_C|}$$
  - If $d_1 \approx d_2$ NNDR ≈ 1 the matching is ambiguous and should be rejected
  - If $d_1 << d_2$ NNDR →0 the matching is correct



Using NNDR for matching, descriptor correctly matches D_A to D_B and correctly rejects matches for D_D.

Feature Tracking

Tracking in Image Sequences



- Key assumptions
  - the intensities of an object point remain the same across frames – brightness constancy
  - the motion of features between consecutive frames is small, and the camera position changes slowly – temporal persistence
  - neighbouring features belong to the same physical surface and have similar motion – spatial coherence

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x,y,t)$$

for simplicity assume to be 1 (i.e., one time step)

Lucas Kanade assumes brightness constancy and temporal persistency.

Lucas-Kanade Algorithm

- The Lucas–Kanade method estimates an optimal solution for point displacements by solving the Least Squares problem:

$$(\mathbf{A}^T\mathbf{A})\mathbf{d} = \mathbf{A}^T\mathbf{B}$$

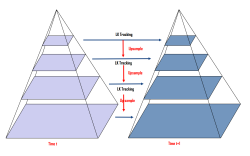
$$\begin{bmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum I_xI_t \\ \sum I_yI_t \end{bmatrix}$$

Have we seen this before?

- The summations are over all pixels in the NxN window where spatial coherence is assumed
- The Lucas–Kanade method estimates an optimal solution for the point displacements by solving the above least squares problem. Rather than using the Harris corner detector, it uses the Shi-Tomasi implementation, and therefore we call this Lucas-Kanade-Tomasi Tracker (LKT).

For larger displacement, we can use multi-resolution on an image.

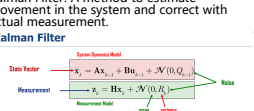
LK Tracking with Image Pyramid



Construct image pyramids, apply LK tracker to low-resolution images, propagate results to higher-resolution images, apply LK tracker ...

Kalman Filter: A method to estimate movement in the system and correct with actual measurement.

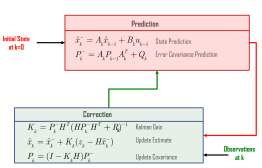
Kalman Filter



x_t – the state vector containing the terms of interest for the objects (position, velocity, heading, acceleration etc);
u_t – the vector containing control inputs (steering, breaking etc);
A – called state transition matrix, determines the effect of each system state parameter at time t-1 on the system state at t (e.g. how velocity and acceleration will affect the position between image frames)
B – the control input matrix (e.g. how external force can affect the movement of the object)
z_t – the measurements we can make from the images
H – the transition matrix that maps the state vectors to the measurement space (e.g. projection of the object features onto the image space)

Kalman Filter

- The Kalman filter estimates the state vectors iteratively by following these two steps:
  - ➢ **Prediction**
    The state estimate and error covariance matrix at time t_t is predicted given the input measurements up to t_{t-1}
  - ➢ **Correction**
    Update the state estimate and error covariance matrices with the observation model



Added non-linear extension to Kalman filters

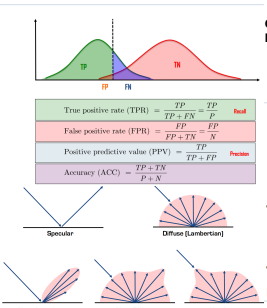
Extended Kalman Filters (EKF)

- Non-linear extension of the original Kalman filter
- The non-linear behaviours of the system's dynamics are approximated by local linearisation around the last state estimate
- Each iteration of the EKF consists of
  - The last filtered state estimate
  - Linearisation of the system dynamics around the last state estimate
  - Prediction of the Kalman filter to the linearised system dynamics just obtained
  - Linearise the observation model around the prediction
  - Apply the filtering or update part of the Kalman filter to the linearised observation model
- It is the EKF that we normally use for vision, particularly for complex scenes (e.g. robotic navigation in surgery)

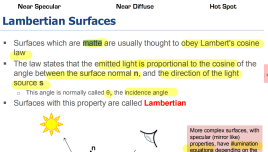
$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) \qquad x_k \approx \overline{x_k} + A(x_{k-1} - \overline{x_{k-1}}) + Ww_{k-1}$$
$$z_k = h(x_k, v_k) \qquad z_k \approx \overline{z_k} + H(x_k - \overline{x_k}) + Vv_k$$

- A is the Jacobian matrix of partial derivatives of f with respect to x
- W is the Jacobian matrix of partial derivatives of f with respect to w
- H is the Jacobian matrix of partial derivatives of h with respect to x
- V is the Jacobian matrix of partial derivatives of h with respect to v

Calculation of Error Rates

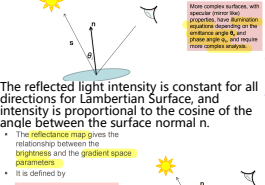


True positive rate (TPR) $= \frac{TP}{TP+FN} = \frac{TP}{P}$	Recall
False positive rate (FPR) $= \frac{FP}{FP+TN} = \frac{FP}{N}$	
Positive predictive value (PPV) $= \frac{TP}{TP+FP}$	Precision
Accuracy (ACC) $= \frac{TP+TN}{P+N}$	



Lambertian Surfaces

- Surfaces which are matte are usually thought to obey Lambert's cosine law
- The law states that the emitted light is proportional to the cosine of the angle between the surface normal **n**, and the direction of the light source **s**
  - This angle is normally called θ_i, the incidence angle
- Surfaces with this property are called **Lambertian**



More complex surfaces, with specular (mirror like) properties, have illumination equations depending on the emittance angle θ_e and phase angle φ_e, and require more complex analysis.

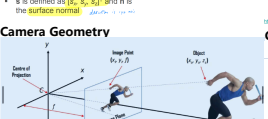
The reflected light intensity is constant for all directions for Lambertian Surface, and intensity is proportional to the cosine of the angle between the surface normal n.

- The reflectance map gives the relationship between the brightness and the gradient space parameters
- It is defined by

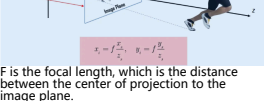
$$R(p,q) = \rho \frac{\mathbf{n} \cdot \mathbf{s}}{|\mathbf{n}||\mathbf{s}|}$$

- ρ is the reflectance factor, or albedo
- **s** is defined as $[\sigma_p \;\; \sigma_q \;\; \sigma_p \overline{p} + \sigma_q \overline{q}]^T$ and n is the surface normal. Derivation (1 - 2 - 3 - 4 - 5)

Camera Geometry

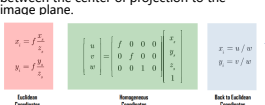


F is the focal length, which is the distance between the center of projection to the image plane.



Internal Coordinates

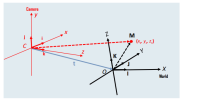
- Let's try to simplify the notation even further



- **K** is called the calibration matrix (3x3 upper triangular matrix), representing the internal parameters of the camera and has five DoF.

K is the intrinsic matrix, the former model is the projection matrix, mapping 2D points to 3D.

Camera Transformation – in homogeneous coordinates



$$F = \begin{bmatrix} a & b & \alpha a + \beta b \\ c & d & \alpha c + \beta d \\ e & f & \alpha e + \beta f \end{bmatrix}$$

- F has 7 degrees of freedom (3x3 – 1 (homogeneous) -1(rank 2))
- If **F** is the fundamental matrix for camera pair (**P**, **P'**), then F^T is for (**P'**, **P**), and $\mathbf{F}^T\mathbf{P}'$ represents the epipolar line for **x'** in the second image
- For any point **x** except **e**, the epipolar line $\mathbf{l'}=\mathbf{Fx}$ contains the epipole **e'**, therefore

$$\mathbf{e}'^T\mathbf{Fx} = (\mathbf{e}'^T\mathbf{Fx}) = 0 \text{ for all } \mathbf{x}, \text{ therefore } \mathbf{e}'^T\mathbf{F} = 0, \text{ or } \mathbf{F}'\mathbf{e}' = 0, \text{ similarly } \mathbf{Fe} = 0$$

Essential Matrix

- When the calibration matrix K is known, we can apply the inverse

$$\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x} \text{ then } \hat{\mathbf{x}} = [\mathbf{R} \,|\, \mathbf{t}]\mathbf{x}$$

- This effectively removes all the intrinsic calibration factors and is called a normalised camera matrix. In this case, we can write

$$\mathbf{P} = [\mathbf{I} \,|\, 0], \quad \mathbf{P}' = [\mathbf{R} \,|\, \mathbf{t}]$$

- The fundamental matrix corresponding to the pair of normalised cameras is called the **Essential Matrix**, which can be written as

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}$$

- The defining equation for the essential matrix is

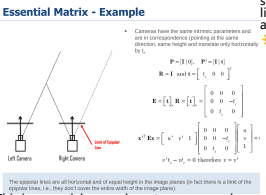
$$\hat{\mathbf{x}}'^T \mathbf{E}\hat{\mathbf{x}} = 0$$

Derivation for the general expression of E, where the camera matrices are not considered K and P are not identity matrices

$$\mathbf{F} = \mathbf{e}'_\times(\mathbf{P}'\mathbf{P}^+)$$
$$= [\mathbf{P}'\mathbf{C}]_\times \mathbf{P}'\mathbf{P}^+$$
$$= [\mathbf{t}']_\times \mathbf{K}'\mathbf{R}\mathbf{K}^{-1}$$

All coordinates are normalized (By times K^-1), then we can simplify the fundamental matrix to the essential matrix.

Essential Matrix - Example



- Cameras have the same intrinsic parameters and are in correspondence (pointing in the same direction, same height and translate only horizontally by t_x

$$\mathbf{P} = [\mathbf{I} \,|\, 0], \quad \mathbf{P}' = [\mathbf{I} \,|\, \mathbf{t}]$$
$$\mathbf{R} = \mathbf{I} \text{ and } \mathbf{t} = \begin{bmatrix} t_x \\ 0 \\ 0 \end{bmatrix}$$
$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R} = [\mathbf{t}]_\times = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_x \\ 0 & t_x & 0 \end{bmatrix}$$
$$\mathbf{x}'^T\mathbf{E}\mathbf{x} = \begin{bmatrix} u' & v' & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_x \\ 0 & t_x & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 1$$
$$v't_x - vt_x = 0 \text{ therefore } v = v'$$

The epipolar lines are all horizontal and of equal height in the image planes (in fact there is a limit of the epipole when t_x tends to zero, i.e. they don't cover the entire width of the image plane).

This is a special case where two cameras have the same camera parameters and are horizontal with the same orientation, making R identity.

Computational Steps for Estimating F

A naïve algorithm for calculating F can be formulated to include the following main steps

- Feature extraction, ensure features are salient and stable
- Calculate a set of potential matches (to begin with you need at least 8)
- Do
  - Select matched feature-pairs
  - Calculate an initial estimation for F from AF=0 using SVD
  - Re-projection of feature points using the estimated F
  - Determine how many points are re-projected with large errors
  - Remove those outliers and look for additional matches
- Until (the number of outliers is less than a pre-set %)
- Refine F based on all correct matches

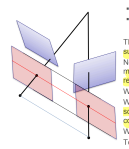
The foundational matrix can also be estimated by solving the constraints by collecting a number of potential matches.

Stereo Constraints and Priors

- Even with the use of epipolar constraints after successful recovery of the fundamental matrix, stereo reconstruction is still a difficult problem as there can still be error prone
- This can be due to the absence of reliable features or structures with a lot repeated structure points, and therefore makes robust matching difficult
- **Uniqueness Constraint** – for any point in one image, there should be at most one matching point in the other image
- **Smoothness Constraint** – disparity values change slowly for most part of the image
- **Ordering Constraint** – the corresponding points should be in the same order in both views after projection

Stereo Image Rectification

- Computationally searching along epipolar lines can be expensive as most of them are obliquely oriented
- Numerical error can also impose problems
- Stereo image rectification is the process of re-projecting image planes onto a common one that is parallel to the base line linking camera centres
- After rectification, all epipolar lines will then become horizontal lines, which makes the search algorithm particularly simple
- The rectification is effectively a simple image warping process, which is easily done by graphics card, particularly modern GPUs
- What's the fundamental matrix for the image plane after rectification?



Rectification makes the two image planes on the same greater plane, making all epipolar lines horizontal, which makes the computation to find matches easier.

- When camera is only rotated but not translated, there is a simple map between the rotated coordinates and the original one
- Original (home) position

$$\mathbf{x} = \mathbf{K}[\mathbf{I} \,|\, 0] \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{X}$$

- After rotation R, we have

$$\mathbf{x}' = \mathbf{K}[\mathbf{R} \,|\, 0] \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{R}\mathbf{X}$$

- Therefore, we have

$$\mathbf{x}' = (\mathbf{K}\mathbf{R}\mathbf{K}^{-1})\mathbf{x}$$

homography

- Once the disparity $x - x'$ is known, the **depth** of the 3D point corresponding to x can be calculated using the equation:

$$d = \frac{tf}{x - x'}$$

Where:

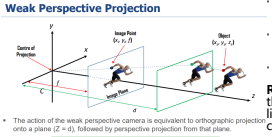
- d: Depth of the point in 3D space.
- t: **Baseline distance**, i.e., the distance between the two camera centers.
- f: **Focal length** of the camera, which is a measure of how strongly the camera converges light.
- $x - x'$: **Disparity**, the horizontal distance between corresponding points in the two images.

Challenges in Stereo Matching

The slide also highlights common challenges that make stereo matching difficult:

1. **Paucity of Salient Features:**
   - Uniform surfaces lack unique features to match between images.
2. **Occlusions and Repetitions:**
   - Occluded points (visible in one image but not in the other) and repetitive textures (e.g., patterns) make it hard to identify unique correspondences.
3. **Specular Highlights:**
   - Reflective surfaces (e.g., shiny objects) create view-dependent artifacts that disrupt matching.

Weak Perspective Projection

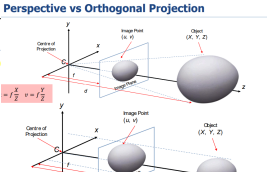


- The action of the weak perspective camera is equivalent to orthographic projection onto a plane (Z = d), followed by perspective projection from that plane.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \alpha_x & s & 0 & 0 \\ 0 & \alpha_y & 0 & 0 \\ 0 & 0 & 0 & d \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Weak perspective projections are accurate when the object is small and distant from the camera.

Perspective vs Orthogonal Projection



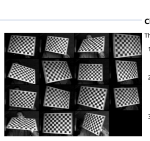
Camera distortion is put as a parameter in Intrinsic parameters.

Camera Calibration

- Intrinsic camera parameters: the principle point, focal length, scaling factors for pixels, skew factor, and distortion factors

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 2D checkerboard patterns are commonly used for camera calibration
- The corners of the squares are used as calibration points with known 2D to 3D correspondences



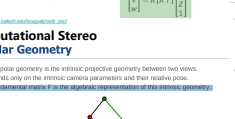
$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = K[R \,|\, T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

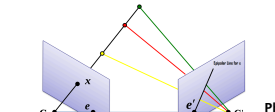
http://www.vision.caltech.edu/bouguetj/calib_doc/

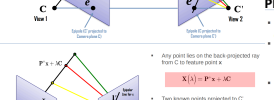
Computational Stereo

Epipolar Geometry

- The epipolar geometry is the intrinsic projective geometry between two views.
- It depends only on the intrinsic camera parameters and their relative pose.
- The fundamental matrix F is the algebraic representation of this intrinsic geometry.







Epipolar constraint: Reduces the correspondence problem to a 1D search along conjugate epipolar lines C'1 and C'2.

Any point lies on the back-projected ray from C to feature point x

$$\mathbf{X}(\lambda) = \mathbf{P}^+\mathbf{x} + \lambda\mathbf{C}$$

Two known points projected to C' coordinate-system x' and x'' (you can choose any point on the ray (other than C of course), e.g., choose λ=0)

$$\mathbf{x}' = \mathbf{P}'(\mathbf{X}(\lambda))$$
$$= \mathbf{P}'(\mathbf{P}^+\mathbf{x} + \lambda\mathbf{C}) = \mathbf{P}'\mathbf{P}^+\mathbf{x}$$
$$\mathbf{e}' = \mathbf{P}'\mathbf{C}$$

Two points in C' space define line l'

$$\mathbf{l}' = \mathbf{e}' \times (\mathbf{P}'\mathbf{P}^+)\mathbf{x}$$

Which yields the fundamental matrix

$$\mathbf{F} = \mathbf{e}' \times (\mathbf{P}'\mathbf{P}^+)$$

Product of two points is a line, product of two lines is their intersection. X(λ) is the 3D point projected from 2D image plane.

- The fundamental matrix satisfies the condition that for any pair of corresponding points x↔x' in the two images

$$\mathbf{x}'^T\mathbf{F}\mathbf{x} = 0$$

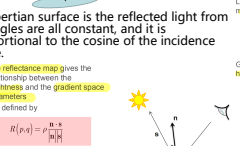
- This is because if x and x' correspond, then x' lies on the epipolar line l'=Fx or, in other words

$$\mathbf{x}' \bullet (\mathbf{F}\mathbf{x}) = 0, \quad \text{or} \quad \mathbf{x}'^T\mathbf{F}\mathbf{x} = 0$$

- Conversely, if the image points satisfy the above condition, then the rays defined by these points are co-planar, which is a necessary condition for points to be in correspondence.

Photometric Stereo

- Surfaces which are matte are usually thought to obey Lambert's cosine law.
- The law states that the emitted light is proportional to the cosine of the angle between the surface normal **n**, and the direction of the light source **s**.
  - This angle is normally called θ_i, the incidence angle
- Surfaces with this property are called **Lambertian**



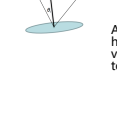
More complex surfaces, with specular (mirror like) properties, have illumination equations depending on the emittance angle θ_e and phase angle φ_e, and require more complex analysis.

Lambertian surface is the reflected light from all angles are all constant, and it is proportional to the cosine of the incidence angle.

- The reflectance map gives the relationship between the brightness and the gradient space parameters
- It is defined by

$$R(p,q) = \rho \frac{\mathbf{n} \cdot \mathbf{s}}{|\mathbf{n}||\mathbf{s}|}$$

- ρ is the reflectance factor, or albedo
- **s** is defined as $[\sigma_p \;\; \sigma_q \;\; \sigma_p \overline{p} + \sigma_q \overline{q}]^T$ and n is the surface normal



As we have the assumption that nearby pixels have the same velocity, thus the same u and v. Then use the least square method, similar to Lucas-Kanade, to solve it and get u and v.

A surface can be parameterized as:

$$\mathbf{X}(X,Y) = (X, Y, f(X,Y))$$

The normal vector of a surface is given as the cross product of the partial derivatives

$$\mathbf{n} = \frac{d\mathbf{S}}{dX} \times \frac{d\mathbf{S}}{dY}$$

$$\left[1, 0, \frac{\partial Z}{\partial X}\right]^T \text{ and } \left[0, 1, \frac{\partial Z}{\partial Y}\right]^T \text{ by defining } p = \left[\frac{\partial Z}{\partial X}\right] \text{ and } q = \left[\frac{\partial Z}{\partial Y}\right]$$

We take derivatives of the surface with respect to X and Y; the two vectors are shown in green. For Photometric Stereo, we have Orthogonal Projection.

- We can define the above slope vectors as $[1, 0, p]^T$ and $[0, 1, q]^T$, the unit normal vector is the cross product of the two, we have therefore

$$\mathbf{n} = [1, 0, -p]^T \times [0, 1, -q]^T = \frac{1}{\sqrt{1+p^2+q^2}}[p, q, 1]^T$$

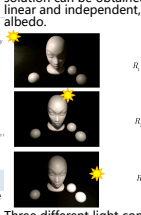
- In vision literature, for convenience, people don't always use the unit normal vector, but rather just to use $[p, q, -1]^T$. Furthermore, we may use the outward normal, in which case it is

$$[\partial Z/\partial X, \partial Z/\partial Y, -1]^T \text{ rather than } [-\partial Z/\partial X, -\partial Z/\partial Y, 1]^T$$

defined above, so read carefully.

- For cases where the light direction is not coincident with the view direction, then for constant contours of R(p,q), the equation is a general second order conic section.
- The maximum of R(p, q) is at (p, q) = $(s_x/s_z, s_y/s_z)$

The above equations are non-linear. A unique solution can be obtained if the equations are linear and independent, and we know the albedo.



$$R_1(p,q) = \rho \frac{s_x p + s_y q + s_z}{\sqrt{1+p^2+q^2}}$$

$$R_2(p,q) = \rho \frac{s_x p + s_y q + s_z}{\sqrt{1+p^2+q^2}}$$

$$R_3(p,q) = \rho \frac{s_x p + s_y q + s_z}{\sqrt{1+p^2+q^2}}$$

Three different light conditions can help us estimate the p and q for every point in the image.

$$\frac{R_1(p,q)}{R_2(p,q)} = \frac{s_x p + s_y q + s_z}{s'_x p + s'_y q + s'_z}$$

$$\frac{R_1(p,q)}{R_3(p,q)} = \frac{s_x p + s_y q + s_z}{s''_x p + s''_y q + s''_z}$$

which is a set of linear equations for each pixel and the unknown albedo is cancelled out. This can be solved easily.

Of course, you can also take different ratios, e.g., R_1/R_3, R_2/R_3 etc.

The method works as straightforward, what are the limitations?

- We now know the p and q value at every pixel, we need to find the depth information from p, q
- This can be done by performing 2D integration, as

$$p = -\frac{\partial Z}{\partial x} \quad \text{(partial derivative)} \qquad q = -\frac{\partial Z}{\partial Y}$$

Note, however, can be problematic in practice because image noise will introduce errors to p and q, and this error will be propagated along the integration path.

After getting P and Q, integration will give us the depth information of the point. However, this is inefficient as we have to do it for every point.

Global Methods of Recovering p, q

- The method previously shown is defined as local, since it calculates each pixel independently
- A different method, called the global method, has been formulated, in which only one light source is used, but a relationship is established between adjacent pixels
- In computer vision, we call this **shape-from-shading**
- It is usually assumed that the objects are smooth, and this smoothness condition is formulated in a relaxation process for estimating (p, q)

The method starts from a pixel (x, y), and it is assumed that an estimate of the surface gradient (p, q) has been made for that pixel

Now, there will, in general, be an error that can be measured between the measured intensity at that pixel I(x, y), and the intensity estimated from the reflectance map R(p, q)

We therefore want to find a process that minimises $(I(x,y) - R(p,q))^2$

We know from the definition of the reflectance map that there is no unique solution to this minimisation problem, and therefore we need to add a further constraint (this is where the smoothness comes in)

We can get a measure of smoothness by taking the derivatives of p and q

To compensate for the differing signs, the smoothness function is taken to be

$$S\left(p_x, p_y, q_x, q_y\right) = \left[\frac{\partial p}{\partial x}\right]^2 + \left[\frac{\partial p}{\partial y}\right]^2 + \left[\frac{\partial q}{\partial x}\right]^2 + \left[\frac{\partial q}{\partial y}\right]^2$$

Thus, we can write an error function to minimise as:

$$E\{p,q,p_x,p_y,q_x,q_y\} = \{I(x,y) - R(p,q)\}^2 + \lambda S(p,q,x,y)$$

where λ is a factor relating the relative importance of the two quantities to be minimised

To solve the global problem we sum the error over the whole image

$$E = \iint E\left(p,q,x,y\right) dx\, dy$$

To find the minimum of the function E we need to differentiate it and set the result to zero (unfortunately this is not a straightforward process, since p and q are derivatives with respect to x and y)

The process is carried out by a method called the calculus of variations, developed by Lagrange and Euler

Limitation of Shape from Shade method:

The light sources must be uniform, or at least a large distance from the object (this is an intrinsic assumption we made when deriving the reflectance equations)

The above formulation also assumes **orthogonal projection**, which for close-up views or objects with large depth variation the assumption will not hold

Another restriction is that surfaces should be **Lambertian**. In practice all surfaces will have a specular content, which will complicate the equations

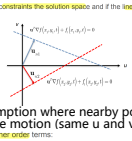
Other formulations have been made in which close point sources of light can be utilised in computer vision.

Optical Flow

- Let's see what happens if two adjacent pixels belonging to the same object moving with the same velocity, we have

$$\begin{bmatrix} I_x(x_1,y_1,t) & I_y(x_1,y_1,t) \\ I_x(x_2,y_2,t) & I_y(x_2,y_2,t) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} I_t(x_1,y_1,t) \\ I_t(x_2,y_2,t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Geometrically, this constraints the solution space and if the lines intersect, we have a solution



Use the assumption where nearby points have the same motion (same u and v).

So, ignoring the higher order terms:

$$\left[\frac{\partial I}{\partial x}\right] dx + \left[\frac{\partial I}{\partial y}\right] dy + \left[\frac{\partial I}{\partial t}\right] dt = 0$$

and using the previous notation for velocity we have that:

$$u = dx/dt \qquad v = dy/dt$$

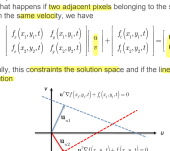
And so we have:

$$\left[\frac{\partial I}{\partial x}\right] u + \left[\frac{\partial I}{\partial y}\right] v + \left[\frac{\partial I}{\partial t}\right] = 0$$

- This is called the **optical flow equation**
- The optical flow assigns to every point on the image a motion vector which represents the velocity at which the point is moving across the image

u and v are 2D velocities of the object, which are also the only two unknowns.

- It is important to note the underlying assumptions for calculating optical flow, which is not always valid
- The figure below shows that a rotating sphere, with fixed light and camera positions will not show any changes of pixel intensities, whereas a sphere that is stationary relative to a moving light source will display intensity changes



Motion but no optical flow

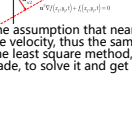
Optical flow but no motion

- Similarly, translation relative to a stationary light source will produce intensity changes. We assume that the lighting is stationary, and consider the movement of rigid objects relative to a fixed camera position

Let's see what happens if two adjacent pixels belonging to the same object moving with the same velocity, we have

$$\begin{bmatrix} I_x(x_1,y_1,t) & I_y(x_1,y_1,t) \\ I_x(x_2,y_2,t) & I_y(x_2,y_2,t) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} I_t(x_1,y_1,t) \\ I_t(x_2,y_2,t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Geometrically, this constraints the solution space and if the lines intersect, we have a solution



As we have the assumption that nearby pixels have the same velocity, thus the same u and v. Then use the least square method, similar to Lucas-Kanade, to solve it and get u and v.

In vision, global techniques are more commonly used. For example, if we assume that we have a measure of the velocity at each pixel (u_i, v_i), then we can estimate the square of the error at each pixel using:

$$R(p_i) = \left[\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\right]^2$$

We can also include a term related to smoothness at the pixel p_i using:

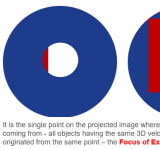
$$S(p_i) = \left[\frac{\partial u}{\partial x}\right]^2 + \left[\frac{\partial u}{\partial y}\right]^2 + \left[\frac{\partial v}{\partial x}\right]^2 + \left[\frac{\partial v}{\partial y}\right]^2$$

These two terms combine into one error equation with the usual parameter λ:

$$E^2(p_i) = R(p_i) + \lambda S(p_i)$$

This can be solved by using calculus of variations

- The **aperture problem** refers to the fact that motion estimation is highly ambiguous when the observation window is very small
- To overcome this problem, use global techniques to calculate optical flow.
- An object corner may support the calculation of the correct 2D motion.



- It is the simple point on the projected image where the object appears to be coming from - all objects having the same 3D velocity in space appear to be originated from the same point – the **Focus of Expansion** (FoE)
- Let's define the velocity components of the moving object (be careful u, v, w are velocities here):

$$u = \frac{\Delta x}{\Delta t}, \; v = \frac{\Delta y}{\Delta t}, \; w = \frac{\Delta z}{\Delta t}$$

- And the perspective projection of a point in the scene onto the camera's image plane:

$$x' = f\frac{x}{z} \qquad y' = f\frac{y}{z}$$

- For simplicity we choose the focal length $f = 1$
- Consider a point at (x_1, y_1, z_1), moving with constant velocity in 3D
- Then at some time interval t later the point will have moved to:

$$x' = \frac{x_1 + ut}{z_1 + wt} \qquad y' = \frac{y_1 + vt}{z_1 + wt}$$

- To find the point where the motion apparently comes from we let $t \to \infty$, which allows us to eliminate $(x_1\ y_1\ z_1)$, to get

$$x' = \frac{u}{w} \qquad y' = \frac{v}{w}$$

- This is the **FoE (focus of expansion)** - a fixed point for movement with constant velocity, irrespective of where the object is
- If the observer changes direction (or objects in the world change their direction), the FOE changes as well.
- The equation allows computation of the relative depths of two points on a moving object:

$$\frac{D_1(t)}{V_1(t)} = \frac{z_1(t)}{w_z(t)} \text{ and } \frac{D_2(t)}{V_2(t)} = \frac{z_2(t)}{w_z(t)} \text{ therefore}$$

$$\frac{D_1(t)V_2(t)}{D_2(t)V_1(t)} = \frac{z_1(t)w_z(t)}{z_2(t)w_z(t)}$$

- and since every point on the body moves with the same velocity in the 3D space, $(w_x(t)=w_y(t))$, therefore

$$z_2(t) = \frac{z_1(t)D_2(t)V_1(t)}{D_1(t)V_2(t)}$$

How to use depth information of one point to estimate the other.

- Once the relative depth is derived, the x and y components can be obtained using the relations:

$$\begin{aligned} z(t) &= w(t)D(t)/V(t) \\ x(t) &= x'(t)z(t) \\ y(t) &= y'(t)z(t) \end{aligned}$$

- so, if one point is known on the body, it is possible to reconstruct the others
- However, this does not result in a practical algorithm, since the ratios D(t)/V(t) cannot be measured with sufficient accuracy in a raster image
- In the above derivation, we assume we can calculate the apparent **image velocity** of the object using optical flow.
- Instead of reconstructing the full 3D trajectory of the object (which can be computationally expensive), this equation allows us to predict collisions **directly from image measurements** $D(t)$ and $V(t)$.

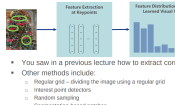
Object Recognition

Challenges in Object Recognition

- Variability within objects
  - View point changes (camera position)
  - Illumination
  - Occlusions
  - Internal camera parameters
  - Scale
  - Deformation
- Variability within class
  - The example of dogs on the previous slide
  - Too many classes

Bag of Features for Image Classification

1. Extract features
2. Learn the 'visual vocabulary' (i.e. The 'dictionary')
3. Quantise the features using the visual vocabulary
4. Represent images by frequencies of 'visual words'



- You saw in a previous lecture how to extract corner or SIFT features
- Other methods include:
  - Regular grid – dividing the image using a regular grid
  - Interest point detectors
  - Random sampling
  - Segmentation-based patches

Normalise patch using the second moment matrix

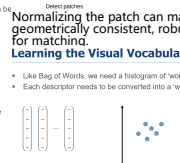
Calculate the descriptor

Detect patches

Normalizing the patch can make it geometrically consistent, robust, and effective for matching.

Learning the Visual Vocabulary

- Like Bag of Words, we need a histogram of 'words'
- Each descriptor needs to be converted into a 'word'



- One method to define 'words' is by using a clustering algorithm such as k-means

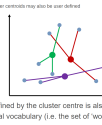
Each descriptor is a point in space; we then use K-means to group them, making them a "word." Then can plot a histogram for each image for their "bag of words."

K-Means Clustering

- One method for performing data clustering
  - K is the number of clusters required and is a user input
  - Minimise the sum of squared Euclidean distances between the points x_i and their nearest cluster centres m_j

$$D(X,M) = \sum_{j=1}^{k} \sum_{i \in j} (x_i - m_k)^2$$

1. The data points are assigned randomly into k groups and the cluster centroids are calculated
   - The initial cluster centroids may also be user defined



- Each 'word' defined by the cluster centre is also known as a **codevector**
- The entire visual vocabulary (i.e. the set of 'words') is also known as a **codebook**
- The codebook can be learned on a separate training set
- The codebook is used for **quantising features**
  - A vector quantiser takes a feature vector and maps it to the index of the nearest codevector in a codebook

- How does one choose vocabulary size?
  - Too small – Visual words are not representative of all patches
  - Too big – Results in overfitting and quantisation artifacts
- There are two machine learning approaches:
  - **Discriminative methods**
    - Learns a decision rule (classifier) assigning bag-of-features representations of images to different classes
    - Examples include Nearest Neighbour, K-Nearest Neighbours, Support Vector Machines, AdaBoost
  - **Generative learning methods**
    - Models the probability of a bag of features given a class
    - Examples include the Naive Bayes classifier or a hierarchical Bayesian models

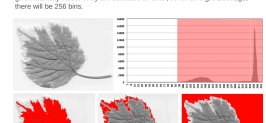
Different Models to classify histogram:

- Nearest neighbour
- K-Nearest Neighbours
- SVM (Linear, Non-linear, and multiclass ones)
  - So for new images:
    - Detect features
    - Classify each feature
    - Examine the frequency of each codeword (compare histograms)

Texture and Region-based Segmentation

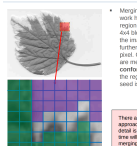
Segmentation by Histogram

- **Histogram**: the frequencies (occurrences) of pixels within the image with a given intensity value. They are allocated to 'bins', so for an eight-bit image, there will be 256 bins.



Histogram methods are vulnerable to noise and lighting effects.

Region Merging

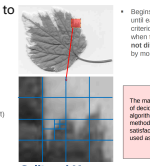


- Merging must start from a uniform seed region. Some work has been done in discovering a suitable seed region. One method is to divide the image into 2x2 or 4x4 blocks and check each one. Another is to divide the image into strips, and then subdivide the strips further. In the worst case the seed will be a single pixel. Once a seed has been found, its neighbours are merged **until no more neighbouring regions conform to the uniformity criterion.** At this point the region is extracted from the image, and a further seed is used to merge another region.

There are some drawbacks which must be noted with this approach. The process is inherently **sequential**, and if fine detail is required in the segmentation then the computing time will be long. Moreover, since in most cases the merging of two regions will change the value of the properties of the region, the resulting segmentation will be dependent on the **search strategy** employed among the neighbours, and the **merging criteria**.

check for uniform criteria and stop after no block satisfies the uniform criteria are found.

Region Splitting

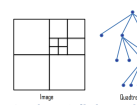


- Begins from the whole image, and divide it up until each sub region is uniform. The usual criterion for stopping the splitting process is when **the properties of a newly split pair do not differ from those of the original region** by more than a threshold.

The main problem with this algorithm is the difficulty of deciding **where to make the partition.** Early algorithms based on histogram segmentation methods and for some classes these are satisfactory; however, in most cases, splitting is used as a first stage of a split/merge algorithm.

Split and Merge

- Region splitting results in many small homogeneous areas, but neighbouring areas may be similar and should be merged.
- Various algorithms combine both processes
  - Either refine a boundary by both techniques, increasing detail as the boundary becomes more accurate
  - Or split image to find large seeds, then merge (usually through regular decomposition)
- An important data structure used in split and merge algorithms is the quadtree



Note that in graphics the quadtree is used in a region splitting algorithm (Warnock's Algorithm) which breaks a graphical image down recursively from the root node, which represents the whole image, to the leaf nodes which each represent a coherent region, which can be rendered without further hidden line elimination calculations. The same use is made of quadtrees for storing

Quadtree Splitting and Merging Algorithm

1. Define an initial segmentation into regions, a homogeneity criterion, and a pyramid data structure.
2. If a region R in the pyramid data structure is not homogeneous, split it into four child-regions; if any of the four regions with the same parent can be merged into a single homogeneous region, merge them. If no region can be split or merged, go to step (3).
3. If there are any two adjacent regions Ri, Rj (even if they are in different pyramid levels or do not have the same parent) that can be merged into a homogeneous region, merge them.
4. Merge small regions with the most similar adjacent region if it is necessary to remove small-size regions.

- What criteria can we use to define thresholds for splitting and merging? How do we define uniformity?
  - Mean
    - Gives no uniformity information ✗
  - Min/Max
    - Works in good images, however random noise can lead to wrong results ✗
  - Variance
    - Is a statistical measure of how close to the mean a set of data is ✓

The measure of mean value along alone gives us no measure of uniformity which is essential in practical cases. Instead we need to use some statistical measure, and it turns out that variance is a convenient, and easy to compute property. Hence at each node in the quadtree we will store both the mean and the variance of the pixels in the subtree.

Fischer's Criterion

- Analysis based on mean and variance makes an assumption that the feature upon which the segmentation is based is **distributed normally**
- Use Fischer's Criterion to discriminate between adjacent areas with differing means and standard deviations

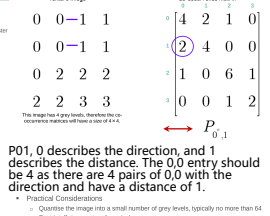
$$\frac{|\mu_1 - \mu_2|}{\sqrt{v_1^2 + v_2^2}} > \lambda$$

- Where λ is a threshold
- If two regions have **good separation in their means** and **low variance**, then we can discriminate them. However, if the variance becomes high and the mean difference is low, it is not possible to separate them.

Split the region into two if the criteria are satisfied.

- Using hue for segmentation has several immediate advantages over intensity
- Shadows: no change in hue, only intensity
- Specular Reflections: no change in hue, only a change in saturation
  - Saturation, like the r, g, and b planes may be helpful in a particular application

Co-occurance Matrices



This image has 4 grey levels, therefore the co-occurrence matrices will have a size of 4×4.

P01, 0 describes the direction, and 1 describes the distance. The 0,0 entry should be 4 as there are 4 pairs of 0,0 with the direction and have a distance of 1.

- Practical Considerations
  - Quantise the image into a small number of grey levels, typically no more than 64
  - Restrict **distances** to < four pixels
  - Restrict the **directions** to four (horizontal, vertical, and two diagonal directions)
    - So from a texture window of any size we would have **sixteen** 64 × 64 matrices

The co-occurrence matrix should be normalized so each entry becomes the probability of co-occurrence. Therefore, it is independent of the window size.

Texture Matching – Other Properties

$$Entropy = -\sum_{a,b} P_{\varphi,d}(a,b) \log\left(P_{\varphi,d}(a,b)\right)$$

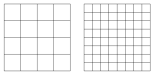
$$Contrast = \sum_{a,b} (a-b)^2 P_{\varphi,d}(a,b)$$

$$Homogeneity = \sum_{a,b} \frac{P_{\varphi,d}(a,b)}{1 + |a-b|}$$

Co-occurrence matrices work well in specific applications but are limited in that they cannot cope with stochastic textures or with textures where the granularity cannot be made to match the pixel size.

Fractal Dimension

- A square may be broken into N self-similar pieces, each with magnification factor r.



$N = 16, r = 1/4$

$N = 64, r = 1/8$

$$D = \frac{\log N}{\log(1/r)}$$

$$D = \log 16/\log 4 = 2\log 4/\log 4 = 2$$

The fractal dimension is a useful feature for texture characterisation although estimation of D from an image is very difficult due to the fact that natural textures do not strictly follow the deterministic repetitive model of fractals assumed above but have statistical variations.