Computer Vision Notes

Vishnu

Tuesday 14th January, 2020

Contents

1 Introduction

Computer vision is the extracting of useful information from images.

1.1 Related Disciplines:

Image Processing Manipulating images

Computer Graphics Synthesising images

Pattern Recognition Recognising and classifying info from datasets/images

Photogrammetry Obtaining measurements (e.g height) from images

Biological Vision Understanding visual perception in humans/animals
(combines neuroscience, psychology and biology)

1.2 Difficulties

Vision exists for specific domains (e.g. the Hawkeye tennis system), but making a general system that works in any environment is an ongoing problem.

1. To a machine, images are an X by Y grid of numbers - even humans would have difficulty
2. Understanding a 2d image of a 3d object is an ill-posed problem
3. A single 3d object can have an exponential number of different images (viewpoints, scale, colour,...)

1.2.1 Ill-Posed Problem

An *ill-posed problem* is one with more than one potential answer/no answer/no direct relationship between the answer and initial conditions.

Converting a 3d image to 2d is well-posed (e.g. looking at a cube head on makes it a square). Converting a 2d to 3d is an inverse problem, so is ill-posed - a single 2d image could have come from multiple different sources (e.g. a square could be one face of a cube or the base of a square pyramid).

By using prior knowledge/context, we can interpret the most likely object for an image.

1.2.2 Exponential images

For any object in an image, it can appear at any position/scale/orientation/colour/deformation/etc. so there are an exponential number of possible images for an object.
Illumination can greatly vary the object's appearance, even if the object itself hasn't changed. There can also be multiple types of the same object (e.g. smartphones), or different classes of objects can look very similar (e.g. calculators and remotes).
In more complex images, multiple objects can occlude/cover parts of each other - this not only deforms the object, but provides lots of background noise.

1.3 Prior Assumptions

Using invariant prior knowledge, we can apply these to images to make better sense of them. The brain has a number of prior assumptions (e.g. the effect of shadows and illumination) that it uses to process images.
Some other priors the brains uses are: perspective, context, prior knowledge and temporal context. We can classify these as:

1. Familiarity - general knowledge. e.g if something is further away, we compensate the fact that it looks smaller

2. Priming - the immediately preceding sensory input. e.g if we just saw a vase on the table, we expect it to be there

3. Context - information about the surrounding scene. e.g. if we see a keyboard and mouse, we expect to see a monitor nearby

There are visual illusions that show us some of the assumptions that human vision makes - these are usually correct, and have been applied to many specialised vision systems.

2 Image Formation

Images are formed when a sensor registers radiation that has interacted with a physical object. e.g. light bouncing off something into your eye

Illuminance Light from source→ object

Luminance Light from object → sensor.

Luminance is a function of the illuminance and the reflectance (Reflective properties) of the object

2.1 Factors

The images formed are affected by two sets of parameters:

1. Radiometric parameters - these determine the intensity/colour of a section of the image. e.g. illuminance type, object reflectance, sensor properties
2. Geometric Parameters - where the 3d points is mapped to in the 2d image. e.g. sensor position/orientation, focal length

2.2 Definitions

Light is electromagnetic waves with wavelength(λ) in the range (400nm,700nm). Human eyes are specialised sensors that are sensitive to this part of the spectrum, and use it for vision. Images can be formed by other waves. e.g infrared cameras, X-ray

Human eyes see different wavelengths as different colours - each source of light emits different amounts of each wavelength, which vary the colour we see them as. *Albedo* is the fraction of light a surface reflects at each wavelength - this gives objects their natural colours. e.g. A ripe tomato reflects red (700nm) well, and doesn't reflect blue (400nm).

Mixing lights is additive - it gives a white light. Mixing pigments is subtractive - we get a black surface (Albedo of 0 for every colour, as it absorbs everything).

Colours aren't a property of the world - the human visual system uses it to represent the albedo of a surface. Determining colour is an ill-posed problem for vision systems, as it depends on illuminance and albedo. Humans are able to naturally determine colours without knowing the exact illuminance.

2.3 Focusing Images

Since light reflects off everything, a sensor would just give us the average illumination. Optics can be used to *focus* light rays onto the sensor to get a useful output.

Focus All rays from a point in the world converge on a single point in the image. This can be done by placing a small aperture between the sensor and the scene (i.e a pinhole camera)

Exposure Time required for the sensor to get enough radiation to measure. Too long gives a blurry image, as the luminance tends to average.

$$Focus \propto \frac{1}{Exposure} \propto \frac{1}{ApertureSize}$$

A larger pinhole gives a brighter image, but makes it blurrier (more average luminance).

2.4 Lenses

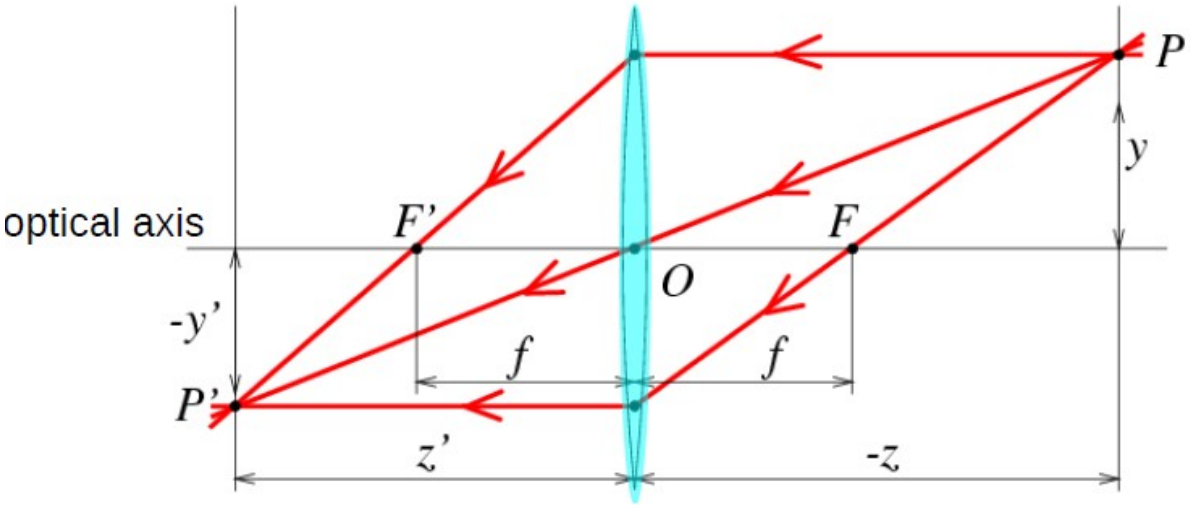
A lens can be used to allow in light, while focusing the scene. It specifically focuses a part of the scene at a specific length from the lens (*focal length*), which somewhat limits it.
A lens is a thin transparent curved piece (usually glass/plastic) that refracts light.

Optical Centre Centre of the lens, light rays that pass through this remain unchanged

Optical Axis The line passing through the optical centre and perpendicular to the lens. Rays that pass parallel to the optical axis pass through the focal point after refraction.

Focal Point The point on the optical axis that represents the focal length of the lens. Rays that pass through the focal length before refraction are parallel to the optical axis after refraction.

In the below image (from the slides): F and F' are the focal points, O is the optical centre, f is the focal length, and the red lines show the three light rays we consider for simple models. Z' and z are the distance of the object and image from the lens respectively.



2.4.1 Thin Lens Equation

$$\frac{1}{f} = \frac{1}{|z|} + \frac{1}{|z'|} \tag{1}$$

Where z is the distance between the lens/object, and z' is the distance between lens/image.

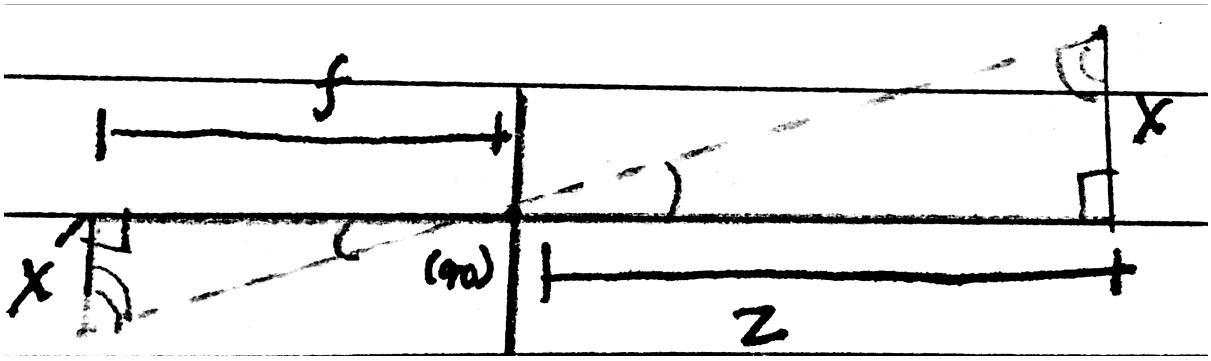
According to the above, objects at infinity will be formed at f - all the scene points will be focused at a single image point, so no meaningful image. For any object closer than f, the light rays won't converge and no image will form. Realistically, anything just beyond 2f and closer will form a meaningful image - the receptor plane is normally placed between (F', 2F').

2.4.2 Focal Range

The *focal range* is the range that an object can be moved without noticing a blur in the image.
The smaller the aperture/lens size, the larger the focal range (but the darker the image, as less light comes in). A pinhole camera has an infinite focal range.

2.5 Perspective/Pinhole camera model

A simple camera model that only considers rays passing through the optical centre, and assumes the image plane is at F'. As a pinhole camera, images are always in focus.



By using similar triangles, we can say that:

$$X' = \frac{f}{z} \cdot X \tag{2}$$

A similar calculation can be done to find the Y co-ordinate of the point, as Z' is fixed at f' (image plane distance).

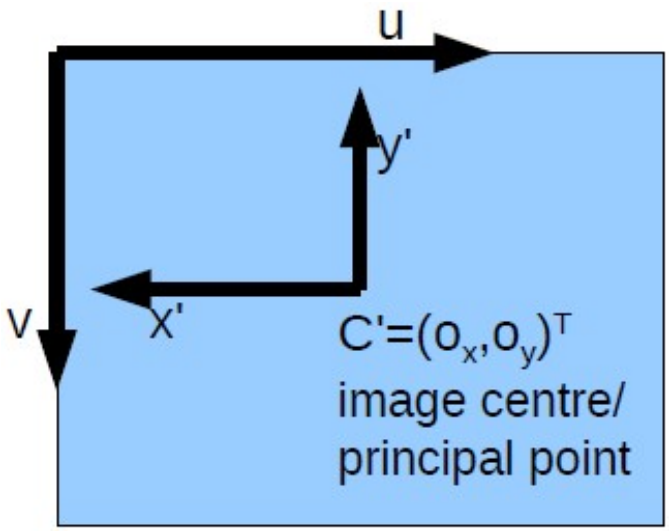
2.5.1 3D to 2D transformation

Applying the above, we obtain the equation:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \frac{f'}{z} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3}$$

$$= \frac{focalLength}{objectDistance} \cdot \text{Projection Operator} \cdot \text{Camera coordinates}$$

The projection operator is determined by the camera model used - for the simple model, it has no effect. The co-ordinates are relative to the camera's optic centre, rather than the image (as seen in the below image from the slides).



2.5.2 Co-ordinate conversion

To change from camera co-ordinates (mm, (0,0) in centre) to image co-ordinates (px, (0,0) in top left corner), we can use the following equations:

$$u = \frac{x'}{\frac{mm}{px}} + \text{Optic Centre}_x = x \cdot \frac{f'}{z} \cdot \frac{1}{S_x} + O_x = \alpha \cdot \frac{x}{z} + O_x$$
$$v = \frac{y'}{\frac{mm}{px}} + \text{Optic Centre}_y = y \cdot \frac{f'}{z} \cdot \frac{1}{S_y} + O_x = \beta \cdot \frac{y}{z} + O_x$$

Where S_x, S_y and f' are constants - $\frac{f'}{S_x} = \alpha$, $\frac{f'}{S_y} = \beta$
and $\frac{\alpha}{\beta}$ is the camera's aspect ratio. Fitting this into our previous equation, we get:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{z} \cdot \begin{pmatrix} \alpha & 0 & O_x \\ 0 & \beta & O_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{4}$$
$$= \text{Camera parameters} \cdot \text{Projection Operator} \cdot \text{Camera coordinates}$$

2.5.3 Scene Co-ordinates

Camera co-ordinates are fine, but it's more useful to have an external fixed reference (scene co-ordinates). To convert, we use the camera's inverse translation and rotation matrices:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} R & t \\ O_3^T & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{5}$$

Where R is a 3 by 3 rotation matrix, t is a 3 by 1 translation matrix, O_3 is a 3 by 3 zeroes matrix, and 1 is a single element. After applying the inverse and combining this with the previous equation, we get the complete transformation as:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{z} \cdot \begin{pmatrix} \alpha & 0 & O_x \\ 0 & \beta & O_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} R^T & -R^T t \\ O_3^T & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{6}$$

$$= \text{Internal camera parameters}(2D \to 2D) \cdot \text{Projection Operator}(3D \to 2D) \cdot \text{External camera parameters}(3D \to 3D) \cdot \text{World coordinates}$$

2.5.4 Results

The above gives us a well-defined equation from 3D to 2D. However, since we're not given Z when doing 2D to 3D, that's an ill-posed problem.

For convenience, we use a *Virtual Image* - this is the right side up on the same size as the object, otherwise the same as the actual image.

2.6 Projective Geometry

Euclidean Geometry Translations/transformations in a 3D world

Projective Geometry Converts from the 3D to the 2D world (translation, transformation, shear, scaling).

Projective doesn't preserve the shape or angle of the objects, and the size depends on the object distance.

2.6.1 Vanishing Point

The *vanishing point* is the projection of the point at infinity in the image. e.g. a long line away from the camera slowly appears to bend towards the focal point in the image

Parallel lines have the same vanishing point. e.g. looking along train tracks. The human visual system uses this to determine sizes (e.g. two similar objects along a vanishing parallel line, even though the more distant one appears smaller we think they're the same size)

2.7 Digital Image Representation

Digital images have discrete samples of a continuous image (Pixelisation)- each sample is called a picture element (pixel). Each pixel is the average of the intensity value around each sampling point .
These average values are represented with discrete values (Quantization).The more values, the more detailed the picture:

Binary Images Each pixel is 0 or 1

Greyscale Each pixel is a single real value (usually 8bit, which has 256 intensities).
- To convert to a binary image, use a threshold function.

Colour RGB Each pixel has 3 real values - one for Red, Green and Blue intensities respectively (usually 24 bits in total per pixel).
- To convert to a greyscale image, take the average of the intensities.

2.8 Charge Coupled Device Camera

A sensor consisting of a semiconductor with a 2D matrix of photo-sensors. The more light incident on the camera, the more charge flows through the conductor ($\propto$ intensity, exposure time). Micro-lenses are fit on top of each sensor to better focus the light.

For colour, we can use a prism to split the incoming light into colours and measure with 3 separate CCD devices (for R, G and B). Alternatively (and more efficiently), we can use a chequered filter over the whole device, so each photo-sensor only accepts a specific colour. A common filter is the *Bayer Mask*, which has twice as many green as red or blue (since humans are most sensitive to green).

As each sensor only accepts one colour, the missing values have to be filled in via *Demosaicing*, which uses interpolation. Some demosaicing methods are:

1. Nearest Neighbour - Copy the closest pixel of the required colour. Fast and inaccurate
2. Bilinear - Take the average of nearby pixels of the required colour. Fast, not so accurate at edges (since less neighbouring pixels)
3. Smooth Hue Transition
4. Edge Directed Interpolation

2.8.1 Smooth Hue Transition

Uses information from neighbouring pixels, rather than the raw values:

Green Just take the average of the raw values of neighbours

Red/Blue

$$\text{Green value of pixel} \cdot \frac{(\frac{(Red/BlueValue)}{(GreenValue)})\text{for each neighbouring pixel}}{\text{Number of neighbouring Red/Blue pixels}} \tag{7}$$

i.e. Green Value x Bilinear interpolation of ratio (Hue) between Red/Blue and Green

2.8.2 Edge-Directed Interpolation

Just taking an average (As in bilinear and smooth hue) will blur edges, so have an additional check to not interpolate across edges (i.e. interpolate where change in value is lowest)

Green For a pixel (x,y):

- $\Delta H = |G(x-1,y) - G(x+1,y)|$, $\Delta V = |G(x,y-1) - G(x,y+1)|$
- If $\Delta H < \Delta V$: $G(x,y) = \frac{G(x-1,y)+G(x+1,y)}{2}$
- If $\Delta H > \Delta V$: $G(x,y) = \frac{G(x,y-1)+G(x,y+1)}{2}$
- If $\Delta H = \Delta V$: $G(x,y) = \frac{G(x-1,y)+G(x+1,y)+G(x,y-1)+G(x,y+1)}{4}$

Red/Blue Same as Smooth Hue transition

2.9 Image Formation in Humans

The human eye consists of:

Cornea The first lens, which has a fixed focal length

Lens The second lens, which can be stretched to adjust the focal length

Iris The aperture, which can adjust to allow more light/better focus

Retina The image plane, which contains millions of photoreceptors that transduce light to electricity.

Fovea The centre of the image, with the best clarity

Blind Spot Where the optic nerve connects to the retina, no vision here

2.9.1 Retina

The retina consists of layers of transparent cells, with the photoreceptors at the back. There are two types of photoreceptors:

Rods High sensitivity, so can work in low light

Cones Lower sensitivity so need bright light, but there are three subtypes that are sensitive to R, G and B wavelengths respectively

The retina has more cones in the fovea (blue << red, <= green), and more rods at the edges (and overall). The fovea has higher resolution, so more processing space in the brain is allocated to it.

2.9.2 Gangilon cells

Gangilon cells collect the information from photoreceptors: near the edges of the eye, 100s of photoreceptors can connect to a single gangilon, leading to lower resolution than the centre. The cells connect and combine to become the optic nerve, and transmit binary signals.

The *Receptive Field* of the gangilon is the area of the retina that the cell receives information from: roughly circular. This is usually split into a circular centre and a surrounding ring.

Gangilons are either on-centre/off-surround or off-centre/on-surround. These send output when the centre/surround is excited, and inhibit when the surround/centre is excited respectively.

On is excited, Off isn't Outputs more 1s

Off is excited, On isn't Outputs more 0s

Both/Neither are excited Outputs a stream of 50/50 1s and 0s

This structure gives better edge detection, as the only 'different' signals are those where an edge has been detected. This doesn't take into account the illumination, only the contrast. As only the cells for strong contrast areas are active, this allows for more efficient encoding as well - less neurons are active. The split between the types is roughly 50/50.

Ganglion cells can be replicated using Difference of Gaussian filters.

2.9.3 Colour Detection

Colour-Opponent gangilon cells look for blending or opposition between fixed pairs of colours (R/G,B/Yellow) and each pair having on/off centre/surround variants. e.g. on-red-centre/off-green-surround, off-blue-centre/on-yellow-surround

They then trigger when certain colours have been found. e.g. Some cells will trigger on Red or Green, but none will trigger for a red/green hybrid

3 Low Level Vision - Artificial

3.1 Convolution

At each position (x,y) of the image I, a new value is created using a weighted sum of the neighbouring pixels. The neighbours are determined using a *mask* H of size (k,m) centred on (x,y).

$$I'(x,y) = I * H = \sum_{k,m} I(x-k, y-m) \cdot H(k,m) \tag{8}$$

The -ve sign shows that the mask has been flipped or rotated: if it wasn't, that would be cross-correlation rather than convolution. Convolution is commutative/associative (so we can apply multiple to one image), cross-correlation isn't.

At the edges of the image, the mask won't completely fit:

1. Only output a cropped image, where the mask has entirely fit (i.e output size will be (x-2k, y-2m)) ('valid' in matlab)
2. Pad the eges of the image with zeroes, so the output image is the same size as the input ('same' in matlab)

3.1.1 Seperable

A convolution is *seperable* if the mask/kernel (H) can be written as the convolution of two row vectors. i.e. $H = h_1 * h_2^T$
The convolution of two vectors is the same as their product ($h_1 * h_2^T = h_2^T x h_1$). Two 1D convolutions (m+n products per pixel) is much faster than 1 2D convolution (m*n products per pixel).

3.2 Masks

A mask/kernel can be though of as a point-spread function: it uses each point's value in the calculation of its neighbours. The result of a convolution can be thought of as the superimposition of masks, each weighted by the image's pixels.

The mask can also work as a template: when the values in the image section and the **rotated** mask are **both** high, then the pixel gets a large value in the convolved image. These pixels show the location of *features* in the image that correspond to the template.

3.3 Smoothing Masks

Spatial Frequency is how often the pixel values vary. i.e the number of pixels to oscillate between light and dark. A high frequency indicates a sudden change, such as an edge.

Smoothing masks blur/smooth an image when applied, bringing the colours closer together and reducing the spatial frequency. The sum of the values in a smoothing mask is 1.

3.3.1 Box Mask

Every value in the mask is equal (i.e each weight is $\frac{1}{size\_of\_mask}$). This sets each pixel to be the mean average of all of its neighbours, with a larger mask giving a more blurred image.

As the box mask/mean filter has the same values throughout, there is a 'hard' edge: every value is the same inside, and outside the weights suddenly drop to 0. This can produce artifacts (small inconsistencies) in the convolved image.

3.3.2 Gaussian Mask

The values gradually fall off from the centre to the edge of the image: the border pixels have low values, so there's no hard edge. In addition to the size, the mask has a *scale* factor (σ, the standard deviation of the gaussian) used to calculate the values for the mask G:

$$G(X,Y) = \frac{1}{2\pi\sigma} \cdot exp\left(\frac{-(x^2+y^2)}{2\sigma^2}\right) \tag{9}$$

As the scale increases, the size of the mask has to increase or the mask will have a hard edge - roughly $size >= 6\sigma$. The larger σ, the blurrier the image and the lower the frequency (less noise).

A 2D gaussian is seperable into:

$$G(X,Y) = \left(\frac{1}{2\pi\sigma} \cdot exp\left(\frac{-(x^2)}{2\sigma^2}\right)\right)\left(\frac{1}{2\pi\sigma} \cdot exp\left(\frac{-(y^2)}{2\sigma^2}\right)\right) \tag{10}$$

3.4 Difference Masks

An image can be regarded as a 3D plane, where the height is based on the intensity. The difference between two points gives the gradient ($\propto$ spatial frequency), which can be obtained using a difference mask (i.e. approximating differentiation).

These masks usually sum to zero, so in areas of no change (i.e not an edge) we get no response. e.g to get the horizontal gradient ($X' = x_2 - x_1$), we use the mask [-1 1], which gives us $-X'$(the mask flips). We can also obtain the change in change in gradient ($X'' = (x_3 - x_2) - (x_2 - x_1)$) with the mask [1 -2 1].

3.4.1 Intensity Discontinuities

These usually occur at edges in the image, which can represent:

1. Boundary between two items
2. Depth Discontinuities. e.g. a cupboard against a wall
3. Orientation Discontinuities. e.g. two faces meeting at the edge of a box
4. Reflectance Discontinuities. i.e. change in surface material
5. Illumination Discontinuities. e.g. shadows (often hinders object recognition)

3.5 Edge Detection

An edge is a region of the image where the gradients have a common direction and a large magnitude. Simple difference masks can detect edges in one direction .e.g [-1 1] can detect a change from dark to light

3.5.1 Laplacian Mask

The laplacian is an omni-directional difference masks, that can detect a gradient in any direction. e.g.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

It is mathematically equivalent to $-(X'' + Y'')$. Due to it's nature, it's very sensitive to noise (a single pixel that is different from its neighbours).

3.5.2 Laplacian of Gaussian/ Difference of Gaussian

A *Laplacian of Gaussian* mask is a convolution of a Gaussian and a Laplacian: the gaussian removes noise, and the laplacian finds the second derivative in all directions. In practice, it can be simulated with a *Difference of Gaussians* (Gaussian with small σ - Gaussian with large σ).

Both give a mask with high centre values, a low value ring, and median border values: these mimic a centre-surround, like a on-centre gangilon cell.

3.5.3 Gaussian Derivative Mask

These masks are a convolution of a gaussian with a simple difference mask (e.g. [-1 1]). They're more robust than LoG/DoG, but only work in one direction (the results from two masks can be easily combined to produce a superior image).

3.5.4 Canny Edge Detector

A popular edge detection method with the following steps:

1. Convolution using Gaussian Derivative masks to obtain x and y gradient values
2. Combines the results to obtain Magnitude ($\sqrt{x^2 + y^2}$) and Direction ($arctan(\frac{y}{x})$) of each gradient
3. Non-maximum suppression - thins the edges down to single pixels (looks at the pixels on each edge, removes if the neighbour in a perpendicular direction to the edge has a higher magnitude)
4. Converts to a binary image using two threshold (>high threshold = 1, <low threshold = 0, >low and adjacent to one over high = 1, other = 0)

3.6 Feature Detection

Masks can be used as templates: so we construct a mask that matches the feature we're looking for, convolve the image, threshold the convolved image, then search the sections of the image around the remaining pixels.
This has two problems:

1. How do we set the threshold other than arbitrarily?
2. If the image is rotate/scaled/deformed/not bright then the mask won't react strongly

Features are often identified by edges: these are heavily dependent on the scale of the image or scale of the gaussian used.

3.6.1 Scale Invariance

To find features at different scales, we can use either different image sizes or different filter sizes. The second is computationally easier: we create an *Image Pyramid* that contains images of different sizes.

These images can be simply created using down/sub-sampling: $n\downarrow(I)(i,j) = I(n*i, n*j)$. e.g. $2\downarrow$ will take every second pixel. This can be done recursively to get smaller images.

3.6.2 Gaussian Pyramid

Simply down-sampling might create an inaccurate image (e.g. $2\downarrow$ on an image with alternating b/w will create an all black image) - an *aliased* image. This can be alleviated by smoothing with a gaussian the image before sampling. e.g.

$$Image_{n+1} = 2\downarrow(Image_n * G_\sigma)$$

This can also be done recursively, as $2\downarrow(2\downarrow(Image_n * G_\sigma) * G_\sigma) = 4\downarrow(Image_n * G_{\sqrt{2}\sigma})$.

3.6.3 Laplacian Pyramid

Rather than convolve to obtain the intensity discontinuities at each stage (i.e a LoG), we can subtract the image with the convolved image at each stage (before subsampling) to obtain a difference of gaussians. i.e.

$$\begin{aligned} G_n &= Image_n * G_\sigma \\ L_n &= Image_n - g_n \\ Image_{n+1} &= 2\downarrow(G_n) \end{aligned}$$

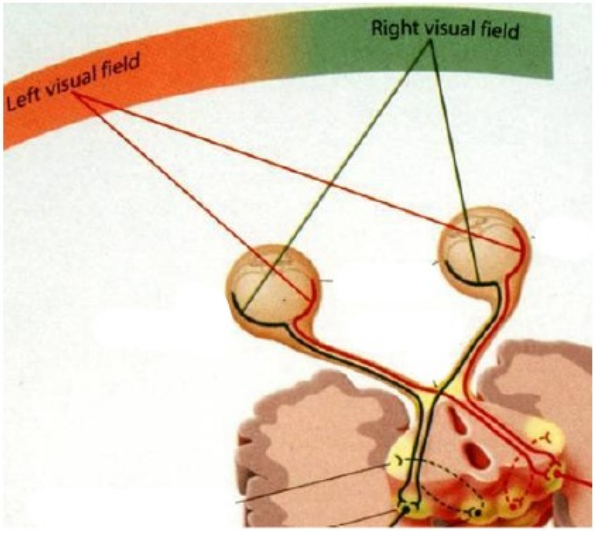
4 Low Level Vision - Biological

4.1 Human Visual System

After the retina, the signals pass through the optic nerve to the *Lateral Geniculate Nucleus*, then to the *Primary Visual Cortex* in the cerebral cortex of the brain.

4.1.1 Left is Right

The right visual field is projected to the left side of each retina and vice versa. Both left sides connect to the left LGN, and both right to the right LGN - so the **rightvisual field** is in the **left LGN**, rather than the left eye (same for the right side).
This can be seen in the below image (taken from the slides):



4.1.2 Lateral Geniculate Nucleus

The section between the optic nerve and the brain. It's mostly considered a relay station consisting of centre-surround cells, but some recent research says that some computation occurs. It's unknown what this computation is.

4.1.3 Cerebral Cortex

All higher processing functions of the brain happen here - 2% of the body mass, 20% of the body's energy consumption. It's divided into specialised areas - there are approx. 30 for vision, approx. 50% of the total volume.

The vision system in the cerebral cortex can be roughly split into two paths:

1. Where - actual vision processing (spatial, colour, motion)
2. What - classification and understanding

Both paths begin in the V1/striate cortex (primary visual cortex, simple/local receptive fields), then move to more specialised areas with larger receptive fields.

4.2 V1 cells

V1 cells have small receptive fields, but have more receptive properties than gangilon/LGN cells, including: orientation, position, direction of motion, colour, binocular disparity and eye of origin.

4.2.1 Colour - Double Opponent cells

Same structure (slightly larger) than centre-surround cells, but specifically look for colour changes/boundaries.e.g. red centre/green surround

4.2.2 Orientation - Simple/Complex/Hyper-complex Cells

Simple V1 cells respond most strongly for a gradient in a particular direction/orientation/ contrast/position in the field - these act as edge/bar detectors.

Complex Cells are like simple, but are more tolerant with contrast/position - still highly orientation selective. These also act as edge detectors, with some flexibility on location.

Hyper-complex cells are like complex, but also depend on the length of the stimulus (length of the edge, not the width). If the stimulus is too long/short, responds weakly. Both complex/hyper-complex are good for motion detection.

4.2.3 Spatial Frequency

Most V1 cells are tuned to a particular spatial frequency (contrast), and respond most strongly to it. Their response drops to almost zero above this frequency, and slowly grows to maximum below it.

4.2.4 Eye of Origin

Monocular Cells only recieve input from one eye: retinal, LGN and some V1 cells.

Binocular Cells receive input from both eyes. They have a separate identical field for each eye, and if both fields have the same response then the cell sends a strong response. This can be used to calculate depth of a stimulus: the disparity (distance) between the stimulus in each eye is inversely proportional to the depth.

4.2.5 Hypercolumnns

V1 cells are grouped into hypercoloumns, based on where in the retina their receptive fields are. Each contains all the cells needed for one section: colour, orientation,etc. in approx $1mm^2$.

The columns are arranged *retinotopically* - adjacent hypercolumns correspond to adjacent sections on the retina. There are more hypercolumns for the fovea, as there are more gangilons/photoreceptors on the fovea.

4.3 Gabor Functions

The receptive fields(RFs) of simple cells act similarly to Gabor Functions (Gaussian x Sinusoid function). These functions have five parameters:

1. σ - Scale of the gaussian - the variance/size of the RF in the mask
2. f - Frequency of the sinusoid - number of alternating regions in the RF
3. Ψ - Phase of the sinusoid - whether the RF starts with an 'on' or 'off' section
4. θ - Orientation of the RF
5. γ - Spatial aspect ratio of the RF

4.3.1 Convolutions

Convolving an image with a single gabor mask gives the results of the simple cell at each position (Each hypercolumn). Convolving with multiple gabor cells (with different phases) and taking the strongest response mimics a complex cell (invariant to contrast) - this can also be obtained with a $\sqrt{sum}$ of two masks with opposing phase (Energy Model).

By varying both the phase and the orientation, we can detect edges in multiple orientations.

4.3.2 Continuous Wavelet Transform

To detect edges at multiple scales, we also have to vary σ. Convolving a signal(image) with similar masks at different frequencies/scales is called a *continuous wavelet transform* - in this case, the wavelet is the gabor.

4.4 Image Components

Images/templates can be thought of as a super-position of various components/elementary features, such as lines or circles. Each component can be weighted (0 if not present), to make any image a weighted sum of a set of components. i.e

$$A \cdot y = x$$

where A is a m by n matrix of image components, y is a n by 1 vector of weights, and x is a m by 1 vector representing a flattened image

This technique can be used to represent multiple images with the same set of components, by making y and x into n by p matrices (p is the number of images). Finding A and y for a set of p images is an ill-posed problem, that can be solved (under certain constraints) by some matrix factorisation methods.

4.4.1 Natural Image Components

Gabor functions work well as natural image components - the brain is able to combine them to represent most images. They're also fairly efficient - most images only weight a few components (*sparsity constraint* to represent images with as few weights as possible), so only a few neurons fire which gives efficient coding.

4.4.2 Applications of Image Components

A similar tactic is used in Jpeg image compression, which uses cosines rather than gabors. It doesn't have a sparsity constraint, but this is still more efficient that storing the whole image. The image is split into patches, each of which is given separate activation weights.

By replicating the image using gabor functions, high frequency data is often lost - this can be used to reduce the noise of the image (*image denoising*). Missing sections of the picture can also be reconstructed by replicating the image (*image inpainting*).

4.5 Non-Classical RFs

The classic receptive field of a neuron is the area of the image/visual field that it responds to. However, in the V1/striate cortex, this response can be affected by the responses from other neurons (context) - the RFs of these neurons are called the non-classical RF.

Co-linear/ Co-circular neighbours (not just adjacent) excite a neuron, parallel neighbours inhibit it. These connections are called the *Assosciation field* - they only enhance/reduce the response, and cannot trigger a response from the neuron if its own RF is empty.

Enhancing/exciting a response makes it easier to see elements that are aligned (smooth contours vs. background noise). Inhibiting similar elements allow outliers to 'pop out' and be easier to see (odd items/boundaries between regions). Both of these contribute to edge detection.

5 Mid Level Vision - Basics and Biological

Low-Level Vision Noise Suppression, edge enhancement, efficient coding

High-Level Vision Object Recognition, Classification, Understanding

Mid-Level Vision Grouping of elements, segmentation/separation of elements from each other

For segmentation, there are two main approaches:

Top-down Segment based on *prior knowledge* of what to look for.
- This is based on familiarity/expectation of what will be there (e.g Pareidolia)

Bottom-up Group based on similarity of appearance/based on *image properties* .e.g Gestalt Laws

5.1 Gestalt Laws

Heuristics (not fixed laws) that guide how elements are grouped in images in humans.

1. Proximity - nearby items are grouped
2. Similarity - same shape,colour,size,orientation,luminance
   - V1 inhibiting connections make outliers 'pop' out
3. Forming a closed contour. e.g. A venn diagram looks like two circles rather than an oval and two 'C's
4. Continuity - elements that form smooth lines or surfaces. e.g a cross is two lines, not two right angles
   - Sometimes we see illusory contours/surfaces to make continuity by *connecting* multiple items together if they're scattered or parts are occluded
   - V1 lateral connections make contours easier to see
5. Common Motion - elements that move together
6. Symmetry e.g. [ ]( ) vs. · ·· ·
7. Common Closed Region - elements already grouped. e.g looking at boxes of items
8. Connectivity. e.g. $*-\cdot\cdot-\diamond$ vs. $*\quad\cdot\cdot\quad\diamond$

The overall abstract principle is *simplicity* - the structure is as simple as possible to describe/remember.

5.2 Object Segmentation

Objects are bounded - boundaries belong to the foreground, so the background doesn't appear to have a border, making it harder to recognise shapes. Only one object can own a border at once, making the other hard to see (Border Ownernship) e.g. the Rubin Face/Vase stimulus

5.2.1 V2 Cells

The V2 cortex (after V1/striate) contains cells that deal with border ownership. They have larger receptive fields, which can check the whole image.

Each region has 2 cells sensitive to a particular orientation (to find the border), with each cell sensitive to one direction. The two neurons compete to determine which side owns the border, using the association field with other V2 cells to determine the probable object. The winning side is the object, the loser is the background.

6 Mid Level Vision - Artificial

6.1 Feature Space

Each location/image section is given a co-ordinate in a *feature space* (e.g. (illumination,colour,size)) and this is used to determine similarity. Objects can then be grouped by:

- Similarity - Affinity, cross-correlation, correlation co-efficient
- Distance - Euclidean, Sum of Square differences, Manhattan

In both of the above, we can weight to prioritise features - determining these weights is non-trivial. Features also have to be scaled appropriately so they're all in the same range/ have equal effect (before weighting) on the output.

6.2 Thresholding

A one-dimensional feature space, the image is segmented into above/below the Threshold ($I'(x,y) = I(x,y) > Threshold$). Brightness is the standard feature, but this can be applied to anything.

6.2.1 Choosing the Threshold

1. Using the average brightness/intensity as the threshold
2. Plot an intensity histogram of the image - these are usually bimodal (two large peaks, that represent the foreground/background respectively), so choose a point between the two peaks
3. Hystersis Thresholding - Use two thresholds (e.g. the 2 peaks in the histogram) with the following conditions:
   (a) $I(x,y) >$ Upper Threshold $\rightarrow I'(x,y) = 1$
   (b) $I(x,y) <$ Lower Threshold $\rightarrow I'(x,y) = 0$
   (c) $I(x,y) <$ Upper Threshold $\&I(x,y) >$Lower Threshold & adjacent to a foreground pixel $\rightarrow I'(x,y) = 1$
   (d) Otherwise $I'(x,y) = 0$
4. Set the Threshold so a certain % of the image is foreground

Usually a single threshold isn't enough - shadows/illumination can vary the average. *Local/Block Thresholding* splits the image into sections (usually quadrants) and uses a separate threshold for each.

Thresholding works well with pre-processed images, as they have a clearly defined feature space. It's a simple method though, and can miss important/include unwanted pixels and sections. It also *doesn't use spatial information* to group segments, which is a large limitation.

6.2.2 Morphological Operations

Operations to clean up the results of thresholding

1. Dilation - expands the foreground to fill in holes/gaps. Every pixel with a foreground neighbour(i.e. has value 1) is set as foreground.
2. Erosion - Reduces the foreground to remove noise. Every pixel that has a background neighbour (i.e. not surrounded by foreground) is set to background (0).
3. Closing - Dilation→Erosion - used to fill in missing pixels
4. Opening - Erosion→Dilation - used to fill in remove noise

6.3 Region-Based Segmentation

Uses a combination of location and feature space to segment images.

6.3.1 Region Growing

1. Select a seed pixel and give it a region label
2. Expand to similar ungrouped neighbouring pixels
3. Repeat 2 until there are no similar pixels left
4. Assign the group its label, then repeat from 1 till no pixels are ungrouped

This method groups similar pixels, but similarity is calculated from the new rather than the seed pixel - slow changes or 'weak' boundaries (intermediate colours) can cause the groups to not segment properly and 'leak' through boundaries.

6.3.2 Region Merging

1. Split the image into regions (every pixel, every 2x2 pixel square, etc.)
2. Select a region, merge with its neighbours if it's similar to create a new region. Repeat this step with the new region
3. If no similar neighbours, mark the region as final
4. Repeat from 2 until all regions are final

This method uses the region as a whole to determine similarity, so is less likely to leak. However, the final regions depend on the selection order - pixels tend to change slowly so lots of intermediate colours, so a region could be similar to both of its neighbours. e.g. [Blue Turquoise Green] could be [B B G] or [B G G]

6.3.3 Region Split and Merge

1. Set the entire image as one region
2. If the region isn't internally consistent (all pixels aren't similar, σ is too high), split into quadrants.
3. Repeat 2 for every region until they're all consistent
4. Apply Region Merging

If the image doesn't cleanly divide by 4, add filler columns/rows and remove after step 3.

6.3.4 Drawbacks

Illumination/shadows heavily affect region merging - a single surface can have variations due to curvature/shadows/lighting.

6.4 Clustering

Groups data points into non-predefined groups/clusters (i.e. unsupervised learning on the feature space). Two main types:

1. Partitional Clustering - divide the data into non-overlapping groups (each data point is in one group)
2. Hierarchical Clustering - the clusters are in a nested hierarchical tree
    - Divisive - start with all the points in one cluster, then recursively split
    - Agglomerative - each point is a separate cluster, and in each iteration merge the most similar clusters

6.4.1 K-Means Clustering

A partitional clustering algorithm that takes as input the number of clusters (k).

0 Choose k data points (image elements in feature space), assign each to be a cluster centre

1. Assign each point to the closest cluster
2. Compute new cluster centres (mean of each cluster's points)
3. If any cluster centre has changed this iteration, loop from 1

This method is heavily dependent on the initial centres, number of clusters, and the distance method chosen (from the cluster centre, from the closest point in each cluster). It also tends to make the clusters the same size, and clusters around a mean point (circle/sphere/etc.), when these aren't always desired..

6.4.2 Agglomerative Clustering

0 Treat each data point as its own cluster

1. Merge the two most 'similar' clusters
2. Repeat 1 until there is a single cluster

The similarity is usually stored in a proximity matrix - the definition of the similarity greatly affects the output:

1. Single Link - the minimum distance between two points in each cluster
2. Complete Link - the maximum distance between two points in each cluster
3. Group Average - the average distance between each point in each cluster
4. Centroid - the distance between the centroid (mean average) of each cluster

6.5 Graph Cutting

The feature space is a fully connected graph (every pair of nodes is connected) with each vertex/node representing a image element. Each edge contains the similarity measure between the two nodes. It's then segmented into subgraphs to group the elements together, by cutting links/edges. These subgraphs have *strong internal links*(high similarity inside the subgraph) and the cut edges are weak.

A cut is a set of vertices removed from the graph to make two distinct subgroups. The 'cost' of a cut for subgraphs (A,V) in a graph (V,E):

$$cut(A.B) = \sum_{a \in A, b \in B} E(a, b) \tag{11}$$

This definition favours cuts with a smaller number of edges, leading to smaller (single element) subgroups. To prevent this, we normalise the costs - for a graph (V,E) and subgraphs (A,B)

$$\begin{aligned} Assoc(A) &= \sum_{a \in A, v \in V - A} s(a, v) \qquad (12) \\ &= \text{total sum of edges coming out of A} \\ NCut(A, B) &= \frac{cut(A, B)}{Assoc(A)} + \frac{cut(A, B)}{Assoc(B)} \qquad (13) \end{aligned}$$

The normalised version tends to favour cuts into equal segments. Finding subgraphs for the minimum cuts is NP-hard, so we normally approximate. The method (like the others) struggles with textured backgrounds.

6.6 Fitting

Using a known mathematical model (e.g. a straight line), try to fit it to a set of image elements. This can be used to find the boundaries of an object, and thus segment the image.

6.6.1 Hough Transform

Rather than trying every possible line/etc. that can fit in an image and seeing how many elements/points it passes through, each point 'votes' for the lines that can pass through it. These votes for each point are stored in an accumulator array and the peaks are chosen.

e.g. a line can be represented as $r = ycos\theta - xsin\theta$ (r is the perpendicular distance of the line to the origin, θ is the forward angle the line makes with the x axis). We can use the points' x and y values to see which (r, θ) pairs pass through each point.
If the points aren't well aligned, then this can results in a number of weak matches and no clearly defined peaks in the accumulator.

6.6.2 Generalised Hough

The above can be easily extended to other shapes. e.g. a circle can be represented as $r = \sqrt{(x-a)^2 + (y-b)^2}$

More generally for shapes that can't be represented with a single equation, they can be described as a location of edge pixels relative to a centre pixel (e.g. $e_1 = (c_x+1, c_y+4), e_2 = (c_x - 4, c_y + 1)$) . For each image element, try each edge position to determine a possible centre pixel - this pixel is then incremented in the accumulator array.

This method is highly tolerant to gaps/occlusion/noise, but is computationally expensive.

6.7 Active Contours/Snakes

Some methods can 'overfit' edges, responding to noise and tiny changes in illumination - we normally want edge contours to be smooth and on the edge.
A 'snake' is a spline curve (a curve defined in pieces by polynomial functions) that tries to fit to an edge by minimising an 'energy' function. This function is a sum of:

1. Shape of the curve - the more bends/length in/of the snake, the higher the energy
2. Proximity - the closer the snake to gradients in the image, the lower the energy

This snake is placed near a gradient, then moves along and closer to it till it fits as a boundary and segments the image. This only works on smooth/closed contours and is very sensitive to the initial position/weighting of the energy sum. The snake moves along the gradient, so if it's not placed near a gradient then it won't move/adjust at all.

6.8 Method Comparisons

Edge-Based partition the image on discontinuities at boundaries
e.g. Hough, active contours, threshholding after edge detection

Region-Based group the image on similarities that make a region
e.g. clustering, region grow/merge/split, graph cutting

Model-based Fit the image to pre-defined models. e.g hough transform, active contours

Model-free Segmentation/Grouping only on the image properties. e.g region grow/merge/split, clustering

7 Correspondence Problem

Given two images, find matching image elements. These images could come from multiple views of the same scene (stereo), the same object at different times (video) or be different and we want to find matching objects. By solving correspondence, we can obtain depth/3D information (from stereo images), perform motion tracking (finding the same object in a video) and object recognition in unrelated images.

This is a search-based problem - for stereo/video the two image as a whole are very similar, which makes solving easier. Some difficulties are:

1. Occlusion - Due to movement, elements in one image might be blocked/have moved out of the other
2. False Matches/Repeating patterns
3. Change in lighting / viewpoint(shear/size/shape)
4. Very large search space - the matching element *could* be anywhere in the other image, but we use priors/knowledge to constrain this

There are two main classes of solution:

Correlation-based Match based on similar image intensities - usually match a 'window' of the image

Feature-based Match based on similar image features. e.g. corners

7.1 Correlation-based methods

Select a region in $Image_1$ and a search space in $Image_2$, then search (using a similarity function) for a match of the region in the search space.

- Window Size - too small and there's not enough detail to match/lots of false matches. Too large, and lower precision/low tolerance to small changes between images
- Search Area Size - too large is computationally expensive, too small could miss the region. Usually centred on the region's location in $Image_1$, but other knowledge can change this (e.g. knowing the exact shift between the two images)

7.1.1 Similarity Methods

A simple approach is minimising the distance between the original and potential regions. i.e Sum of square differences, Euclidean, Manhattan. Manhattan is usually chosen, as the extra computation from the other methods isn't usually worth it.
The other is to maximise the normalised cross correlation. If we consider the regions as vectors I_1 and I_2 in feature space, then the cross-correlation is equivalent to the dot

product. The normalised cross-correlation is then equal to $cos\theta$ (with θ as the angle between the vectors), as shown below:

$$\text{Cross-Correlation} = \sum_{x,y} I_1(x,y)I_2(x,y) \qquad = \|I_1\|\|I_2\|cos\theta \tag{14}$$

$$\text{Normalised Cross-Correlation} = \frac{\sum_{x,y} I_1(x,y)I_2(x,y)}{\sqrt{\sum_{x,y} I_1(x,y)^2}\sqrt{\sum_{x,y} I_2(x,y)^2}} \quad = \frac{\|I_1\|\|I_2\|cos\theta}{\|I_1\|\|I_2\|} = cos\theta \tag{15}$$

$$\text{Correlation coefficient} = \frac{\sum_{x,y}(I_1(x,y) - \bar{I_1})(I_2(x,y) - \bar{I_2})}{\sqrt{\sum_{x,y}(I_1(x,y) - \bar{I_1})^2}\sqrt{\sum_{x,y}(I_2(x,y) - \bar{I_2})^2}} \tag{16}$$

7.2 Advantages/Disadvantages

+ Easy to implement, checks correspondence for any/all points

- Computationally expensive, false positives on repetitive patterns, doesn't work well with change in viewpoint/illumination/size/shape

7.3 Feature-Based Methods

Identify interest points in both images, then match these points by looping through both sets. This only searches interesting points and is less sensitive to change, but doesn't work without 'interesting' points. e.g textured or random regions.

7.3.1 Detecting Interest Points

Any detection method needs to be *repeatable*(finds the same points in both images - invariant to scaling/rotation/translation/illumination changes) and *distinctive*(the feature descriptions are different enough that false matches are low).

Corners are good: they have a steep intensity gradient in multiple directions (any point on an edge looks like any other point).

7.3.2 Corner Detection

1. Generate I_x and I_y - intensity gradients for the image in X/Y directions using a gaussian mask

2. Generate the *Hessian Matrix (H)* for each pixel (sum the gradients in a small window (size of the gaussian))

$$H = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \tag{17}$$

3. Find the eigenvalues (λ_1, λ_2) of H (correspond to the max. slope intensities)

4. Locations with **both** large eigenvalues are corners (1 large is an edge, neither are flat areas)

This works well, but generating eigenvalues is slow.

7.3.3 Harris Corner Detector

Rather than calculating eigenvalues, this computes:

$$\begin{aligned} R(Hessian) &= determinant(H) - k \cdot trace(H) \\ &= (I_x^2 \cdot I_y^2 - (I_x I_y)^2) - k \cdot (\sum I_x^2 + \sum I_y^2) \end{aligned} \tag{18}$$

The trace of a matrix is the sum of the leading diagonal. k has been found to work best at 0.04-0.06.

R(H) is large for corners, negative for edges, and low for flat areas. Large values of R tend to cluster, so the local maxima is taken as the corner. Alternatively, using *non-maximum suppression*, set every pixel with a lower value of R than its neighbour to 0).

R is rotation/illumination invariant, but not scale invariant (e.g. zooming on a curved edge gives multiple corners). This can be solved using an image pyramid (Harris-Laplacian, take the max. neighbour across scales) or use SIFT features.

7.3.4 Scale Invariant Feature Transform (SIFT)

1. Use Difference of Gaussians across multiple image scales. i.e. Laplacian pyramid

2. Detect local maxima/minima across scales (i.e lower/higher than neighbours and neighbours/self in the scale above and below)

3. Filter points, keep those with a high contrast (gradient change)

4. Keep points with sufficient structure: similar to calculating R, but instead keep points with:

$$\frac{(trace(H))^2}{determinant(H)} < \frac{(r+1)^2}{r} \tag{19}$$

r = 10 has been found to work well.

7.3.5 Describing Interest Points

After Harris/SIFT, the points need to be described. Harris simply takes a small region around the point (like correlation methods) - good for translation, not for rotation/illumination/scale changes.
The SIFT algorithm creates a more robust descriptor as below:

1. Designate a region around the interest point (usually 16x16 pixels) at the scale it was found, and apply smoothing

2. Find the orientation/magnitude of gradients of pixels in the region

   For a pixel P x_1 is the pixel to the right, y_1 is the pixel above,
   x_2 the pixel to the left, y_2 the pixel below

$$Magnitude = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$Orientation = tan^{-1}\left(\frac{y_1 - y_2}{x_1 - x_2}\right)$$

3. Create a histogram of all the orientations (placed in 8 orientation bins), weighted by the magnitude/distance to the central pixel

4. Rotate the region until the largest orientation points 'up'

5. Create histograms for the new orientations in sub quadrants (Each 4 by 4 pixels), weighted by magnitude/distance to central pixel
   - 4x4 quadrants, 8 orientation bins per quadrant - 128 element vector to store all the histograms

6. Normalise the vector and use as a descriptor

This is robust, but not perfect - each descriptor can have multiple *putative matches* (potential solutions). *Outliers* are incorrect matches, and *inliers* are correct.

7.4 Calculating Correspondence

To estimate the transformation between images, we can use the matching feature points. However, they're not perfect - to find the most likely transform (max. inliers), we can use the RANSAC algorithm (RANdom SAmpling and Consensus).

7.4.1 RANSAC

The algorithm takes in a set of putative matches (inliers and outliers), and determines the best transform to fit the data.

1. Choose a random minimal subset of matches

2. Calculate a model, using this subset, to explain the shift (i.e $\Delta X, \Delta Y, rotation$)

3. Test the model using the other matches (apply, and use a threshold to see if the model correctly predicts the match's location), and count the number of matches that agree with the model (*consensus set*).

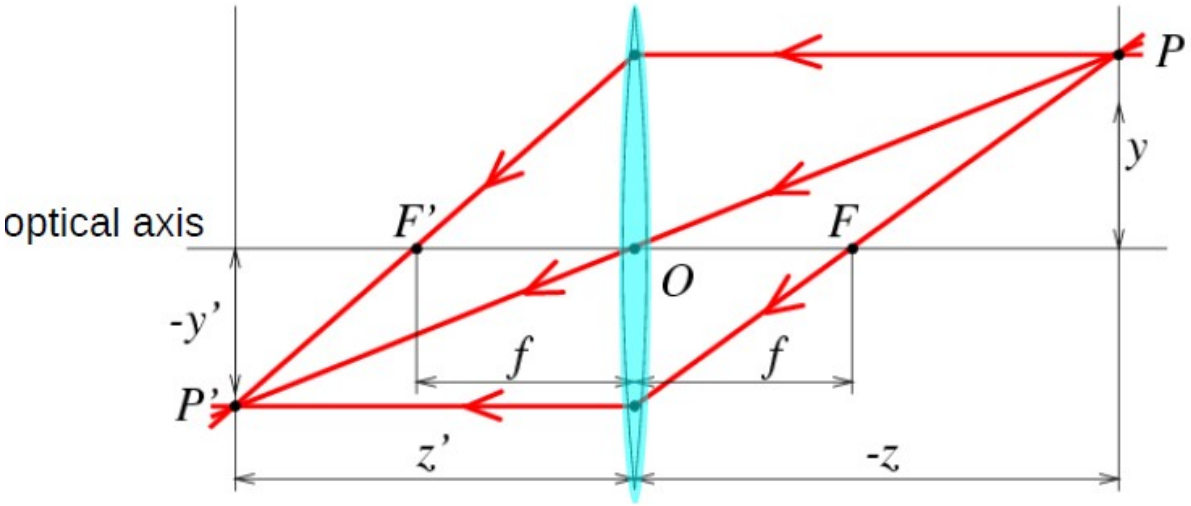
4. Repeat 1-4 N times

5. Choose the model with the largest consensus set

For a simple translation $(\Delta X, \Delta Y)$, one match is enough to generate a model. For rotation, 4 matches are required. This can also be used for line fitting (two points to generate a line model), or other model generation tasks.
It's a simple and general method, but can require lots of iterations and determining N/threshold is difficult.

8 Stereo Vision

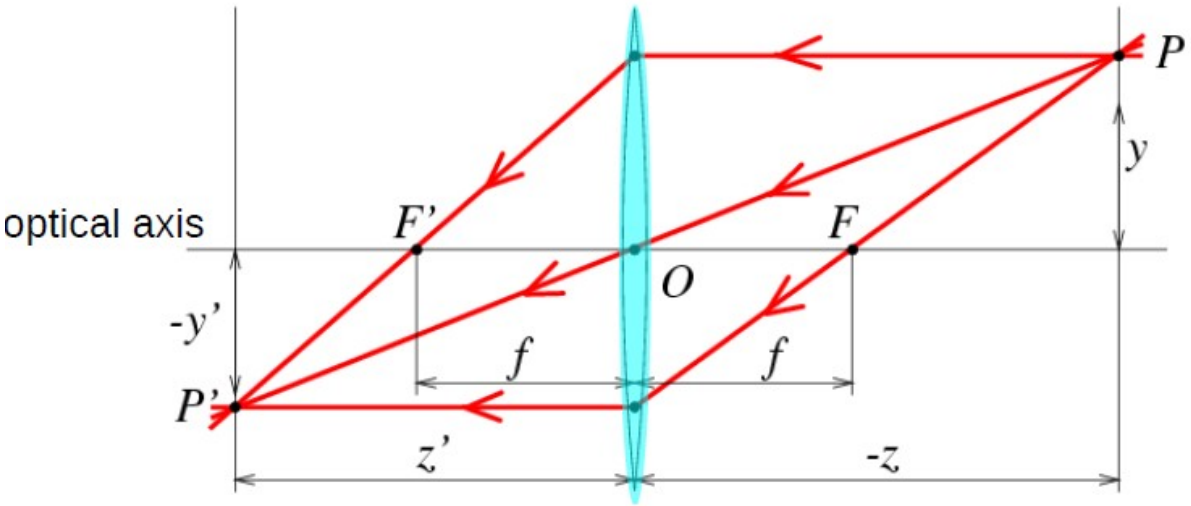
As detailed in section 2.5, when converting from 3D to 2D we lose the depth information. However, with two images + some geometry we can potentially recover this information, as seen in the below image:



By recovering depth information, we can use this to segment objects or recover/record the 3D shape of an object.

8.1 Co-Planar Cameras

Two cameras with co-planar image planes (represented in front of the camera for convenience), same focal length, at the same height and parallel optical axes:



The images are formed on the virtual planes at (x_L, y_L) and (x_R, y_R) for the left/right images respectively. Using the camera model equations (and assuming the image co-ordinate axes have (0,0) in the centre for convenience), we obtain the following for a

scene point (x,y,z) (using O_L as a reference):

$$
\begin{aligned}
x_L &= f_L \cdot \frac{x}{z} && (f_L = f_R) \\
y_L &= f_L \cdot \frac{y}{z} = y_R && \text{(The image planes are at the same height, so } y_L = y_R\text{)} \\
x_R &= f_L \cdot \frac{(x - B)}{z} && \dagger \\
Disparity = d = x_L - x_R &= f_L \cdot \frac{x}{z} - && f_L \cdot \frac{(x - B)}{z} = f \cdot \frac{B}{z}
\end{aligned} \tag{20}
$$

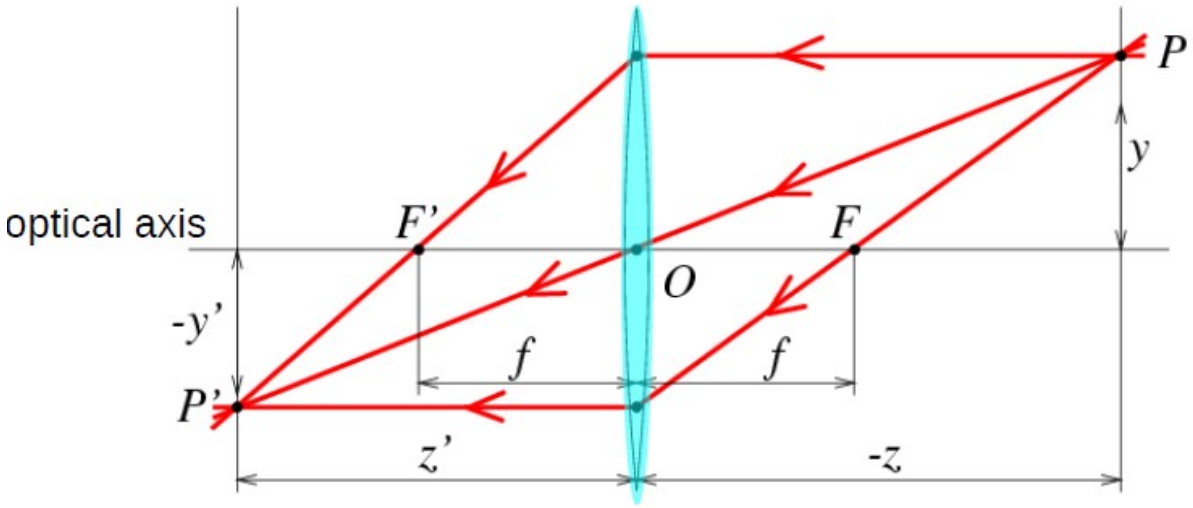
† B is the x-difference between the two cameras. As (x,y,z) is in terms of O_L, B is subtracted to convert to O_R's co-ordinate system.

From the above system of equations, we can see that depth$\propto \frac{1}{disparity}$ - B is known, d can be observed form the images, so z (depth) can be calculated given the two images. If B isn't known, then we can still calculate relative depth. d is simply the difference vector between two corresponding points in the images (horizontal for coplanar cameras).

8.2 Stereo Correspondence

If z (depth) >> B(baseline camera distance), then most points will be visible in both images and corresponding points will be similar - it's still not an easy problem, but there are several constraints we can make use of:

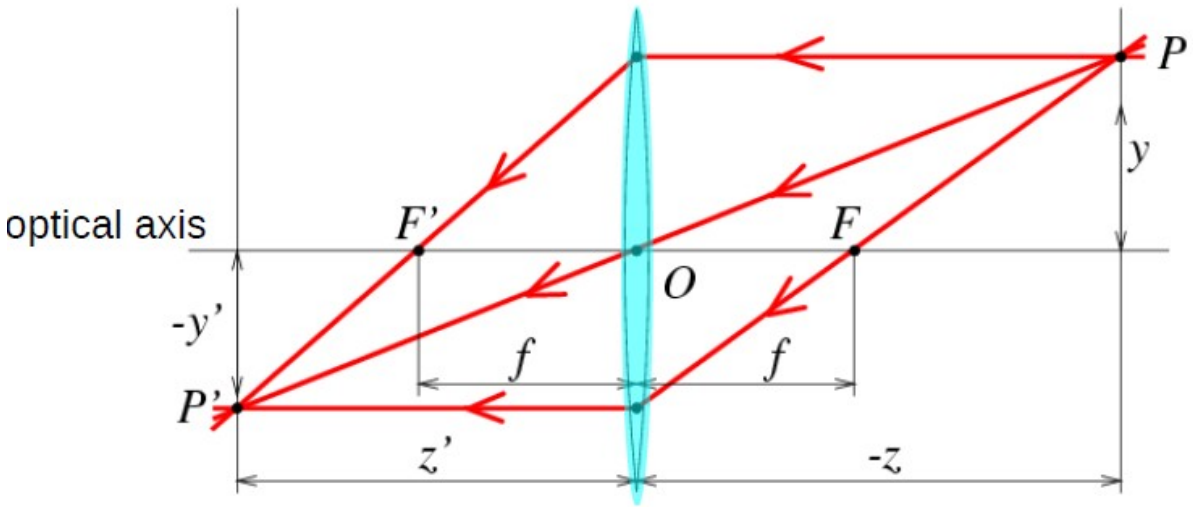
1. Epipolar Constraint - for co-planar cameras, the y-coordinate is the same so it's reduced to a 1D search
2. $Disparity_{max} = f \cdot \frac{B}{Depth_{min}}$
3. Neighbouring points along continuous surfaces will have similar disparities. At sudden discontinuities of depth, this no longer holds
4. Guaranteed Uniqueness - Each point can only map to one point in the other image. The exception is a surface inclined along the line of sight for one camera (e.g. a cuboid directly pointing at one camera) - this will have one point in this camera's image, but multiple in the other camera
5. Ordering - Matching points on *Epipolar Lines* (corresponding co-planar lines, in this case lines with the same y axis) will be in the same order - this doesn't work at depth discontinuities.



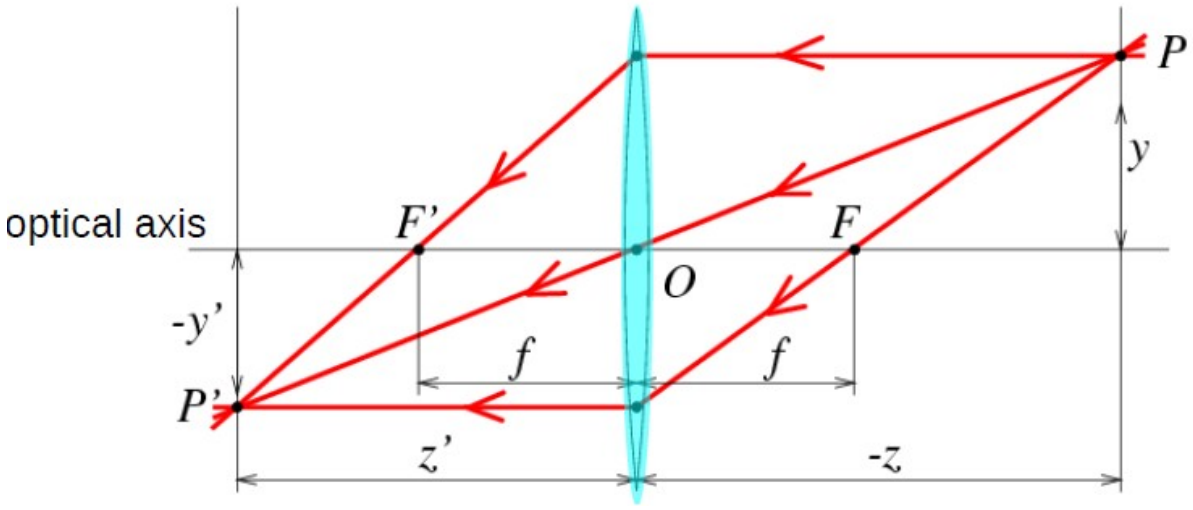
Sometimes we can't find corresponding points due to occlusion or different Fields Of View (depth can only be recovered where the FoV overlap). The closer the cameras, the larger the overlapping FoV, but the larger error in depth estimates ($depth \propto \frac{1}{baselineDistance}$).

8.3 Non-Coplanar Cameras

The cameras/image planes are angled towards each other - the *vergence angle* (θ) is the angle between the optical axes. Disparity is measured using angles rather than distance ($\alpha_L - \alpha_R$) - it's zero at the *fixation point*(where the optical axes meet) and where rays at equal angles intercept (This curve is called the *horopter)*, > 0 further than the horopter and < 0 closer than the horopter.



Eyes are an example of non-coplanar cameras that can adjust the vergence angle. The fixation point is projected to the fovea, points on the horopter are at an equal distance from the fovea in each eye, points further fall outside the fovea on both sides (crossed) and points closer fall inside on both side (uncrossed). Some neurons are tuned to disparity, and can calculate the depth of image points.



8.3.1 Epipolar Geometry

Though the image planes are no longer co-planar, corresponding points still appear on the same lines (though no longer horizontal). These epipolar lines are the intersection of the *epipolar plane*(plane passing through a scene point and both optic centers) with the image planes. *Conjugated Epipolar lines* are the epipolar lines generated from the same epipolar plane - corresponding points always lie on conjugated epipolar lines.

All epipolar lines meet the *baseline* (line between the two camera centres) at the *epipole*(intersection of the image plane with the baseline)(projection of the centre of one camera in the image plane of the other camera, nor usually shown).

Rectification is adjusting the images to make the epipolar lines horizontal.

8.3.2 Calculating the Depth

Unlike in co-planar cameras, there is no simple formula. Depth can be calculated using triangulation (determining the intersection of corresponding view lines) or rectifying the image then using the regular formula - both require knowledge of internal and external camera parameters).

8.4 Depth Cues

8.4.1 Binocular (two images)

Disparity The difference in position of an object between the two images

Accommodation In the eye, how much the lens has been adjusted (to obtain f)

Convergence The vergence angle - known for cameras, can be calculated from muscle movements for eyes

8.4.2 Monocular (One Image)

Interposition If an object occludes another, we assume it's in front of it (uses grouping cues/closed contours to see if it's infront or simply an odd shape)

Size Familiarity If we see two objects identical other than size, then we assume the smaller is further away

Texture Gradients Textures get closer together/smaller with depth

Linear Perspective Parallel lines converge at infinity - these can provide cues to depth/size of objects

Aerial Perspective Light scatters as it travels, so distant objects are blurred/less colourful (Reduced contrast/saturation)

Shading position of light/shadows gives depth cues. The brain assumes light comes from above, so objects with longer shadows must be taller

8.4.3 Motion-Induced

Motion Parallax Speed/direction of image motion with camera/eye movement. Objects closer than the fixation point move in the *opposite direction*, objects further move faster

Optic Flow As the observer moves forwards/backwards, the image expands/contracts. The closer the point the the camera, the faster it moves.

Accretion/Deletion Parts of the image appear/disappear (due to occlusion) as the observer moves, which gives depth cues

Kinetic Depth Movement of the object can induce a 3D perception

9 Video

Video is a series of N images acquired at time intervals ΔT. The intensity of a pixel is represented as I(x,y,t), where t is the time. Video comes from a single camera at different times, stereo images come from multiple cameras at the same time.

9.1 Motion analysis

The projection of a scene onto the image changes with movement in the scene (*object motion*) and movement of the camera(*ego motion*). The *optic flow* is the **apparent** movement of a scene point in the image (i.e. position at time t - position at time $t - \Delta T$) - similar to the disparity in stereo images. The optic flow field can be for specific objects (sparse) or for every point in the image (dense).

In contrast, the *motion field* is the **actual** movement of scene points - this is not always equal to the optic flow. e.g. a for a barber's pole, the optic field is vertical but the motion field is horizontal. The motion field must be estimated from the optic flow.

9.2 Video Correspondence

To find the optic flow, we find corresponding points between two frames and track the motion of objects. Similar to stereo correspondence, there are some constraints we can use:

1. As ΔT is normally small, optic flow vectors will be small
2. Spatial Coherence - assuming smooth surfaces, neighbouring flow vectors will be similar
3. As the viewpoint doesn't change much, most points will be common between the images and will look similar

9.2.1 Aperture Problem

With no change in intensity, it's hard to determine the optic flow (across an intensity gradient is easy, parallel to it is nearly impossible as every point looks the same across an edge). Local motion along the brightness pattern is hard to discern, so the brain usually takes the average/slowest movement (which is movement perpendicular to the pattern).

To solve this, we can combine motion from multiple locations into a global motion, or use locations where the direction isn't ambiguous (e.g. corners - SIFT/Harris)

9.3 Calculating Depth from Optic Flow and Ego Motion

In the following, it is assumed that the object is still and the camera is moving in a roughly straight line. As we calculated earlier, $imageSize = \frac{focalLength}{objectDistance} \cdot objectSize$

9.3.1 Ego Motion ⊥ Optic Axis

The velocity of the camera is V_x, and z is constant. X is the object size, x is the image size.

$$\begin{aligned} z = f \cdot \frac{X_1}{x_1} = f \cdot \frac{X_2}{x_2} = f \cdot \frac{(X_1 - V_x \cdot t)}{x_2} \\ X_1 x_2 = x_1 (X_1 - V_x \cdot t) \\ X_1 (x_2 - x_1) = -V_x \cdot t \cdot x_1 \\ X_1 = \frac{-V_x \cdot x_1}{\frac{(x_2 - x_1)}{t}} = \frac{-V_x \cdot x_1}{\dot{x}} \end{aligned}$$

Note that the object size doesn't change - $X_2 = X_1 - V_x$ w.r.t the initial co-ordinate frame.Plugging this value of X_1 into our original formula, we obtain:

$$Z = f \cdot \frac{X_1}{x_1} = f \cdot \frac{-V_x \cdot x_1}{x_1 \cdot \dot{x}} = -f \cdot \frac{V_x}{\dot{x}} \tag{21}$$

Therefore if $\dot{x}$(change in x in image) can be measured, z(depth) can be recovered.

9.3.2 Ego Motion ∥ Optic Axis

The velocity of the camera is V_z, towards the object (i.e distance to object reduces). Unlike the previous scenario, X has the same value in both camera co-ordinate frames (x still changes).

$$\begin{aligned} fX = x_1 \cdot Z_1 = x_2 \cdot Z_2 \\ x_1 \cdot (Z_2 + v_Z \cdot t) = x_2 \cdot Z_2 \\ X_1 \cdot v_Z \cdot t = Z_2 (x_2 - x_1) \\ Z_2 = x_1 \cdot \frac{V_z}{\dot{x}} \end{aligned} \tag{22}$$

9.4 Calculations from Optic Flow

9.4.1 Time to Collision

In cases where the camera velocity is unknown, we can use the above formula as:

$$\text{Time to collision} = \frac{\text{distance}}{\text{time}} \frac{Z_2}{V_z} = \frac{x_1}{\dot{x}} \tag{23}$$

This assumes V_z is constant, and can be derived solely from the image. Alternatively:

$$\text{Time to collision} = \frac{x_1}{\dot{x}} = \frac{\alpha_1}{\dot{\alpha}} = \frac{2A_1}{\dot{A}} \tag{24}$$

Where α is the angle subtended by the object on the image, and A is the area of the object in the image - these can be obtained without any camera/object information, just the image.

9.4.2 Ego Motion

Optic Flow Vectors are Parallel 1. $V_z = 0$

2. Ego motion direction is the opposite of the optic flow field motion direction
3. Speed of ego motion $\propto$ length of flow vectors

Optic Flow Vectors are Radial 1. $V_z \neq 0$

2. The radial point (where all the vectors point to/away from) is the vanishing point
3. If the vectors point towards the vanishing point, then ego motion is away from the vanishing point (And vice versa)

9.4.3 Relative Depth

Optic Flow Vectors are Parallel Depth of a point $\propto$ (magnitude of flow vector)$^{-1}$ (i.e. motion parallax)

Optic Flow Vectors are Radial Depth of a point $\propto$ (magnitude of flow vector)$^{-1}$ $\propto$ Distance from point to vanishing point

9.4.4 Segmentation

Discontinuities in the optic flow field indicate a difference in movement: this could be the foreground vs. background, or two different objects.

9.5 Tracking

If optic flow methods are applied to high-level objects rather than pixels, it's called *tracking*. Most algorithms use previous frames to predict the location of the object in the next frame: this constricts the search space. Some common methods are particle/kalman filtering. Humans tend to predict as well, using previous data + prior knowledge of how objects usually move.

9.5.1 Segmentation

Motion leads to segmentation. i.e. Gestalt law of common fate

A simple way to track motion is *image differencing*: for a static camera, threshhold the difference between subsequent images to find locations where motion has occurred (i.e. $Image_\delta = (I(x, y, t+1) - I(x, y, t)) > threshold$. This works well to find the boundaries of a moving object, but not internal motion (e.g. if a white square moves from left to right on a black background, the image intensities only change at the left/right edges of the square - differencing will show two rectangles moving, rather than one square).

9.5.2 Background Subtraction

To counter this, we instead take the difference between the frame and the static parts of the scene (the background). This is still messy and requires morphological operations to clean up the image, but shows the entire object + objects that are stationary for a few frames. There are a few methods to select the background:

1. The previous frame. i.e. image differencing
2. A pre-computed background (e.g. take a video of the empty scene and average the frames to get a background - *Off-line Average*)
3. Moving Average - the background B is a decaying weighted sum of previous frames ($B(x, y) = \beta I(x, y, t) + (1 - \beta)B(x, y)$) . This method is most commonly used, as it overcomes slow changes in illumination.

Ideally the background will contain illumination changes/ changes to static background objects, and allow us to easily identify interesting moving objects (even if they temporarily stop).

10 High Level Vision - Artificial

Object Recognition consists of:

1. Classification - Determine what category an object is (e.g. phone vs calculator)
2. Identification - Determine the specific identity of an object (e.g. Pikachu vs Mickey Mouse)
3. Localisation - Determine where/if the object is in the image

If localisation is accurate enough, it can act as *semantic segmentation* - split the image on categories.

10.1 Category Hierarchy

Classification places a scene into a category - an object can be part of multiple categories simultaneously. Identification is classification at the lowest category level. Some categories are subsets of others:

$$\text{Superordinate Categories} \begin{cases} object \\ animal/building \\ mammal/reptile \end{cases}$$
$$\text{Basic level} \{ Dog/Cat/Cow$$
$$\text{Subordinate categories} \begin{cases} Labrador/Poodle/ \\ Golden/Black/Brown/ \\ (Specificnames) \end{cases}$$

The basic level is the fastest for humans, the first one that children learn to do, and usually the step humans perform before identification. It's the highest level at which members look similar, and the lowest where there are easy methods of discrimination.

10.2 Requirements

Object recognition needs to deal with background noise, viewpoint/lighting changes, non-rigid deformations and intra-category variation (e.g. two chairs can look very different while still being chairs). To achieve this it uses image data, representations of objects and matching techniques.

The standard method creates object representations from standard/training images, then matches these representations to the target image.

10.3 Template Matching

Given a template of the object, match it against every image region to calculate the similarity, then threshold the peaks to find the object locations. Similarity methods are as in 7.1.1: maximise cross-correlation (this can be found using convolution), minimise the distance (euclidean/manhattan/sum of squares).

10.3.1 Problems

The method is inherently not robust to change: the target needs to be very similar to the template to be able to match. We could use multiple templates to account for change in rotation/illumination/etc., but this increases the number of false matches. Raising the threshold to reduce false matches will also miss many true matches, as the method isn't very robust.

By adding multiple templates, and checking on every region at multiple scales, the method soon becomes very intractable. It's also highly sensitive to occlusion.

10.4 Sliding Window

Rather than a template, this method runs patches of the image through a classifier (such as a CNN) - it takes patches of different size/shape and resizes them as appropriate (which covers multiple scales and increases tolerance).

Classifying every region of every size/shape/location is impossible, so it instead segments the image to find regions that might contain an object, and processes only those.

10.5 Edge Matching

Similar to template matching, but pre-processes the template/image to extract edges first. It then finds the location with the minimum distance between the location and the template.

Distance between image/template = avg. of minimum distance between each point of the template and a point in the image

10.6 Model-Based Recognition

Make a model/template, convert to edges, then rotate/scale/etc. to find occurances at each patch of the image. This method can work for 2D and 3D models, but just using edges is unreliable on textured images. To combat this, it also looks for the relative orientations of connected edges - this still isn't very good.

10.7 Intensity Histograms

Create a histogram of the intensities/colours in the template and each image region, then compare the two. This method is very fast, and insensitive to small changes - so is often used as part of face recognition, as the skin has a standard number of colours so can be

used to heavily contract the search space.
On the downside, it's sensitive to illumination and doesn't preserve spatial ordering. e.g. a chessboard and a zebra could have the same intensity histogram.

10.8 Implicit Shape Model

Represents the template with two components: the actual parts/image features, and the structure the parts should follow. This allows flexibility in the exact arrangement of the parts, as long as the overall structure is followed - it searches the image for the parts and checks if they're in an acceptable configuration.

10.8.1 Preprocessing/Representation

1. Find interest points for the object in the training images (e.g. Harris,SIFT)
2. Extract image patches around the points from the whole training image dataset
3. Cluster the patches, the centroid of each patch is added to a *Appearance Codebook*
4. Template match the codebook to each training image, and determine where the centre of the object should be relative to each of the codebook features (determination through a process like the generalised Hough transform (Each feature votes for where the centre could be))

10.8.2 Matching

1. Find the interest points in the image
2. Match these points to the code book features
3. Determine the object centre (again via the generalised hough transform)
4. Back-project the codebook features from the centres to highlight where the image matches

10.9 Feature-Based Recognition

Extract robust local features (such as SIFT) from the template objects and convert these to descriptors. Perform the same extraction on the image, and compare against the template descriptors. For this method to work, the descriptors need to be invariant to change in size/translation/rotation/lighting, as well as consistent to find and sufficiently dense.

While matching, the system takes the top two descriptors that match the image and checks the distance ratio between them (distance between the image and the template). If the distance ratio is below a certain threshold, that means both descriptors are almost equally likely - this isn't very discriminating, and may have been a random match so is discarded.

10.9.1 Model Fitting

If three matching non-collinear points can be found, then a translation model can be created to match the rest of the template to the image. This model can be obtained using RANSAC/Hough Transform/etc.

10.10 Bag of Words

10.10.1 For Words

1. Documents are parsed into words
2. Common words (e.g. the, and) are removed, and the remaining words are converted to base forms (e.g. ran,running→ run)
3. Each word is given a unique numerical identifier
4. A histogram (ordered by identifier) is created, then converted into a string and finally a vector

To match with another document, it's also converted into a vector then the angle between them is computed (i.e cos^{-1}{Normalised cross-correlation} (as in 7.1.1))

10.10.2 For Images

1. Select image features/regions (all, interest points, randomly)
2. Encode the features using a descriptor (e.g. SIFT descriptor)
3. Take the features from across the dataset and cluster them - each cluster is part of a *visual codeword dictionary*
4. The most common codewords (appear in most/all images) are removed, as they're too generic to be useful

Images can then be compared to training images by computing visual codeword histograms for both and finding the angle between them.

10.11 Geometric Invariants

A property of the scene that don't change with a change in viewpoint. What invariants are available depend on the type of viewpoint change:

Similarity Space (translation/rotation/scale) ratio of lengths and angles are invariant

Affine Space (similarity + sheer) parallel lines, ratio of areas and ratio of lengths along lines are invariant

Projective Space (affine + foreshortening) cross-ratio (ratio of ratios of lengths along lines)

10.12 Local vs. Global Representation

10.12.1 Local

Objects are broken down into simple features. e.g. A = / + \+ -, T = - + |
This is tolerant to changes in viewpoint/occlusion/variation within category, but doesn't preserve spatial ordering and can potentially match multiple objects. e.g. T = - + |, L = - + |

10.12.2 Global

The overall shape of the image is used. This distinguishes well between similar images, but is sensitive to viewpoint changes/occlusion/variation

11 High Level Vision - Artificial

11.1 Prior Representations

There are two main hypotheses on how humans store visual information in the brain:

1. Object-Based - The brain stores 3D models of images
2. Image-Based - the brain stores 2D views of the object from multiple viewpoints

11.1.1 Object Based

The representations are stored as *Structural Descriptions* - the model is segmented into standard parts and stored with the relationships between the parts.

The theory says that any object can be represented using a standard set of geometrical shapes called *Geons*(geometrical icons). e.g. cube, sphere, cylinder, cone. Geons are sufficiently different from each other, robust to noise/occlusion and look similar from all viewpoints - this gives a robust method of identification from any viewpoint.

To match an image, simply deconstruct it in the same way and compare the description with existing structure representations.

The problems with this are:

1. Decomposing an image is very difficult (we've not yet mastered segmentation)
2. Hard to accurately represent some models. e.g. a tree
3. Doesn't keep fine details, so some representations would be every similar. e.g. cats vs dogs

11.1.2 Image Based

This method is quite similar to template matching. i.e. nowhere near robust enough to explain the human visual system. More recent theories suggest that multiple templates are stored, and interpolations between these are used to match against new viewpoints/slight differencces in the object.

11.2 Processing Images

The two main theories on how humans process images are:

1. Configural - process the image as a whole
2. Featural - process individual features, ignoring the relationships between them

It's been shown that humans use both, depending on the situation.

11.3 Classifying New Objects

There are three main theories on how humans set classification boundaries and classify new images: Rule-based, Protoype-based and Exemplar-based. In all, the new image is converted into a feature space that can be used to describe it then categorised. Distance measures in feature space are the usual, as described in 7.1.1 (maximise normalised cross-correlation, minimise distance).

11.3.1 Rules

Each category is described with a set of rules. e.g. triangles have three sides, birds have a beak/feathers/can fly. New objects are sorted into categories based on these rules.

For Humans are good at extracting rules from nature. e.g. children pick up on the rules of grammar and apply them before they've learnt the exceptions

Against Graded Membership - some things are 'better' members of a class than others (e.g. a bear is a better mammal than a blue whale). If we only use rules, this shouldn't exist - everything it just a member.

11.3.2 Prototype

The centroid/average/prototype of each category is calculated, and new images are given the label of the 'closest' prototype.

For Humans are quick at identifying new images if they resemble old ones.

Against If categories are only represented by the prototypes in the brain, we shouldn't be able to remember/see variations

11.3.3 Exemplar

Specific individual instances (exemplars) are stored in each category - new images are given the label of the closest exemplar.

For Humans are quick at identifying new images if they resemble old ones.

Against Graded Membership

11.3.4 Supervised Learning

Prototype/Exemplar classification are the same as standard ML classification methods in supervised learning (where we have a set of known data points). They correspond to Nearest Mean Classifier and Nearest Neighbour Classification respectively - NMC only works with linear boundaries between classes, and NNC is susceptible to outliers/incorrect data.

K-nearest neighbours is similar to exemplar, but takes the majority label from the nearest K neighbours. K is normally small and odd to prevent ties, but large enough to prevent a single outlier from affecting the outcome.

11.4 Cortical Visual System

11.4.1 Receptive Field Hierarchy

The further along the visual system (from gangilons→LGN cells→V1 cells), the larger the receptive field and the more tolerant to location the neurons are. The stimuli the cells respond to also get progressively more complex (e.g. gangilons are centre-surround, simple V1 cells respond to lines, V2 cells perform border ownership...)

11.4.2 Feedforward Model

The *feedforward model* states that neurons receive inputs from lower neurons to build more complex representations they respond to. As they receive inputs from multiple cells in different locations, they become progressively more tolerant to location. There are two main feedforward models: HMAX and CNN

11.4.3 HMAX

Neurons alternate in layers of *simple* and *complex* cells (similar to V1 cells).

Simple (S) Cells these perform SUM/AND operations, combining inputs from multiple cells to create a more complex shape. e.g. combining two lines to respond to a cross. They take information from the layer below.

Complex (C) Cells these perform MAX/OR operations, combining inputs to receive information from a large area. e.g. accepting any line at a particular orientation, irrespective of position in the image. They take information from multiple simple cells.

Neurons closer to the eye (lower) create a feature vector built from standard image components - this output is generic, reusable and hard-wired. Higher neurons do specific classifications on objects. i.e. supervised learning classifiers

11.4.4 Convolutional Neural Network

These are similar to HMAX, but use standard image processing techniques

Convolution (C) Layer Apply different convolutions to the image to extract features - like simple cells in HMAX

Subsampling (S) Layer Shrink the feature maps by taking the max. from each group of indices - increases tolerance to location as it takes the highest from each. Similar to complex cells in HMAX

11.4.5 Recurrent Connections

In the brain, it's been found that neuron layers/regions also send information to those at the same layer (lateral connections) and the layer below (feedback connections). These allow top-down and bottom-up information to be combined.

Bottom-up Using information from just the stimulus

Top-down Uses context + prior information/knowledge

To combine top-down/bottom-up information in a mathematical way, we can use Bayes' Theorem.

11.5 Bayes' Theorem

$$P(A|B) = P(B|A) \cdot \frac{P(B)}{P(A)} \tag{25}$$
$$\text{Posterior} = \text{Likelihood} \cdot \frac{\text{Prior}}{\text{Evidence}}$$

For vision, we want to know the probability of a given image representing an object - $P(object|image)$. This is an inverse/ill-posed problem, so it's hard to solve. $P(image|object)$ is a forward problem - still very hard to solve, but possible. We can use Bayes' theorem to convert from one to the other:

$$P(O|I) = P(I|O) \cdot \frac{P(O)}{P(I)} \tag{26}$$

We can set $P(I)$ to 1, as we already have the image/evidence and we're trying to determine the object - $P(I/O)$ is geometric (what's the likelihood that the object produces that image, so based on the current information) and $P(O)$ is the probability of seeing that object (e.g. in a city, we're not likely to see a zebra - based on prior knowledge).
We can loop through this for all possible objects - and use various constraints to limit this to a reasonable set of objects.

11.5.1 Single Object Inference

If we only want to know if the image contains a particular object or not, we instead can determine the ratio of seeing/not seeing the object:

$$\frac{P(O|I)}{P(\overline{O}|I)} = \frac{P(I|O)}{P(\overline{I}|O)} \cdot \frac{P(O)}{P(\overline{O})} \tag{27}$$

If this ratio is > 1, then the object is present.

Lots of systems use priors to constrain the search space or make assumptions about the image - Bayesian Inference gives us a mathematical way to use these.