Image Processing and Computer Vision - Notes

Dom Hutchinson

November 26, 2019

Contents

1   Image Acquistion

Proposition 1.1 - *Common Challenges with Image Acquistion*
Below are some common challeges that are faced/produced by image acquistion

Viewpoint Variation	Several images may be taken of the same object but will vary the angle
Illumination	Images may be taken in low/high light
Occlusion	Object may be partly obscured
Scale	Objects may look vary different when placed next to other objects due to their relative scale
Deformation	Objects may have slight variations on the perfect form
Background Clutter	Lots happening behind an object may work to obscure it
Object Intra-Class Variation	Some objects in the same class can vary a lot in shape (*e.g.* chairs)
Local Ambiguity	Certain regions of an image can be missinturpred without the rest of the scene being accounted for
World Behind the Image	Depth may need to be accounted for to make sense of an image.

Definition 1.1 - *Dirac Delta-Function, δ*
The *Dirac Delta-Function* is used to map continuous distributions to discrete distributions by
sampling at particular intervals. Intuitively

$$\delta(t) = \begin{cases} 1 & ,t = 0 \\ 0 & ,t \neq 0 \end{cases} \implies \delta(t - \alpha) = \begin{cases} 1 & ,t = \alpha \\ 0 & ,t \neq \alpha \end{cases}$$

Definition 1.2 - *Sifting Property*
We can apply the *Dirac Delta-Function* to a function to sample a particular value

$$\int_{-\infty}^{\infty} f(t)\delta(t)dt = f(0) \implies \int_{-\infty}^{\infty} f(t)\delta(t - \alpha)dt = f(\alpha)$$

This can be applied to 2D objects (such as images) as

$$\int_{-\infty}^{\infty} f(a,b)\delta(a - x, b - y)dadb = f(x, y)$$

Definition 1.3 - *Point Spread-Function*
A *Point Spread-Function* is applied after sampling an image. It takes the value of a pixel &
transforms pixels around it using this value in some way.

e.g.   (Should be a white dot on black background but ink).

2   Image Representation

Definition 2.1 - *Colour Space*
Colour Space are different techniques for representing colours. These are generally made up of
3D vectors.

Colour Space	Vector Description
RGB	(Red $\in [0, 255]$,Green $\in [0, 255]$,Blue $\in [0, 255]$)
HSI	(Hue $\in [0, 360)$,Saturation $\in [0, 1]$,Intensity $\in [0, 1]$) Hue gives the colour in degrees
YUV	(Brightness $\in [0, 255]$,Blue Projection $\in [0, 255]$,Red Projection $\in [0, 255]$)
La*b*	(Luminance $\in [0, 100]$, Red/Green $\in \{-a, +b\}$, Blue/Yellow $\in \{+b, -b\}$

Remark 2.1 - *Representing Video*
To represent video each fixel is given a third parameter, *time* so we now have

$$f(x, y, t) \mapsto (R, G, B)$$

or any other *Colour Space*.

Definition 2.2 - *Quantisation*
Quantisation is representing a continuous single channel function with discrete single channel function that groups the continuous values into a set number of levels.

Example 2.1 - *Quantisation*

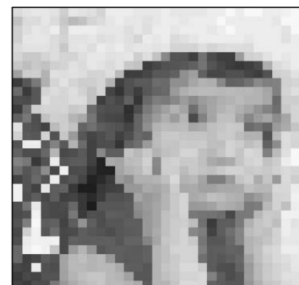| 16 levels | 6 levels | 2 levels |

Definition 2.3 - *Aliasing*
Aliasing is the result of sparse sampling since single pixels represent to large an area to get any detail out of it.

Example 2.2 - *Aliasing*

| 256 x256 | 64x64 | 32x32 |

Definition 2.4 - *Anti-Aliasing*
Anti-Aliasing is the process for avoiding *Aliasing*. This can be achieved by using a sampling rate which is a critical limit defined by the *Shannon-Nyquist Theorem*.

Theorem 2.1 - *Shannon-Nyquist Theorem*
An analogue signal with maximum frequency xHz may be completley reconstructed if regular samples are taken with frequency $2x$Hz.

Definition 2.5 - *Convolution*
Convolution is an operation which takes two functions & produces a third which describes how the shape of one of the two functions is changed by the other.
For functions f & g

$$(f * g)(x) := \int_{-\infty}^{\infty} f(x - t)h(x)\partial t$$

N.B. $*$ is the symbol for convolution.

Remark 2.2 - *Convolution in Image Representation*
Suppose you have a system, represented by kernel $g(x)$, & an input signal, represented by $f(x)$. THen $f * g(x)$ describes the effect of the system on the input signal. The resulting image is called the *Response of f to the kernel h.*

Proposition 2.1 - *2D Discrete Convolution*
Since images are represnted by discrete 2D functions $f : \mathbb{N} \times \mathbb{N} \to (\mathbb{N} \times \mathbb{N} \times \mathbb{N})$ it is pertinent to understand *2D Discrete Convolution.*

$$h(x,y) = \sum_{i \in I} \sum_{j \in J} f(x-i, y-j) g(i,j)$$

Often the kernel, $g(x,y)$, has negative indices so the pixel being acted upto is equivalent to the middle pixel in the matrix representation of $g(x,y)$.
N.B. A convolution whose kernel is symmetric on 180 degree rotation is called a *Correlation.*

Example 2.3 - *2D Discrete Convolution*
Below is a representation of a grayscale image, $f(x,y)$, on the left & a kernel $g(x,y)$ on the

	$y-1$	y	$y+1$	
$x-1$	43	12	61	
x	44	45	60	
$x+1$	43	50	61	

	-1	0	1
-1	-1	0	1
0	-2	0	2
1	-1	0	1

right.
$(f * g)(x,y) = f(x+1, y+1)g(-1,-1) + f(x+1, y)g(0-1, 0) + \cdots + f(x-1, y-1)g(1,1) = -68.$

Example 2.4 - *Kernels*
Kernels an be defined with specific outcomes in mind.

Operation	Matrix
Identity	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
Edge Detection	$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}$
	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 1 \end{pmatrix}$
	$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$
Sharpen	$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$
Box Blur	$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
Gaussian Blur 3×3	$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$

Gaussian Blur 5×5	$\dfrac{1}{256}$	$\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 5 & 4 & 1 \end{pmatrix}$
Unsharp Masking 5×5	$\dfrac{-1}{256}$	$\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$

3   Frequency Domains & Image Transforms

Definition 3.1 - *Image Transform*
An *Image Transform* is deriving a new representation of the input data by encoding the image using another parameter space (*e.g.* Fourier, DCT, Wavelet, etc.).

Remark 3.1 - *Purpose of Image Transforms*
Image Transforms can be used in

  i) Image Filtering;

 ii) Image Compression;

iii) Feature Extraction;

 iv) etc.

Definition 3.2 - *Properties of a Signal*
A *Signal* is a sinusoidal function over continuous time. They have the following properties

  i) Frequency - Number of cycles per second, Hz;

 ii) Period - Length of a cycle, s;

iii) Amplitude - Peak intensity of the signal;

 iv) Phase - The shift of the trig wave from its default position, π.

Theorem 3.1 - *Fourier's Theorem*
All periodic functions over continous time can be expressed as a sum of sin &cos terms, each with their own amplitude & shift.

$$f(t) = a_0 \sin(t + \theta_0) + a_1 \cos(t + \theta_1) + \dots$$

Example 3.1 - *Fourier Transform*
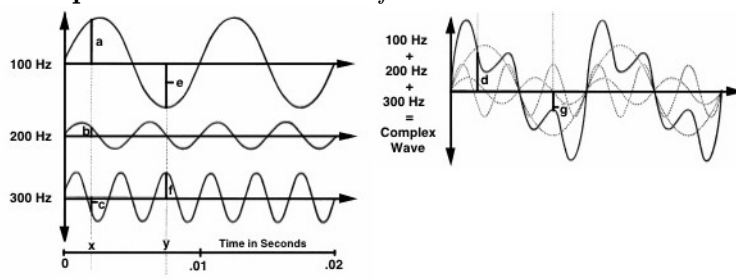
Proposition 3.1 - *Frequency in Images*
Frequency in Images is measured as the rate of change in intensity along a given line on the image.

Remark 3.2 - *Fourier Transform on Frequency in Images*
If we read the intensity values along a single row or column we can produce a sinusoidal wave which generalises the distribution & then perform a *Fourier Transform.*

Definition 3.3 - *2D Discrete-Space Fourier Transform*
Images can be considered as 2-Dimensional discrete space. Let $f(x, y)$ be the intensity of the pixel at position (x, y). 2D Discrete-Space Fourier Transforms have two variables: $u \in [-\pi, \pi)$ for the vertical frequency; and, $v \in [-\pi, \pi)$ for the horizontal frequency.

$$\underbrace{F(u, v)}_{\text{Fourier Space}} = \sum_{y=0}^{m-1} \sum_{x=0}^{n-1} f(x, y) e^{i(ux + vy)}$$

$$= \sum_{y=0}^{m-1} \sum_{x=0}^{n-1} f(x, y) \left[\cos(ux + vy) + \sin(ux + vy) \right]$$

N.B. $F(u, v)$ is a complex number.

Proposition 3.2 - *Interpretations of 2D Fourier Transform*
Since $F(u, v)$ is a complex number we cannot plot it exactly. Thus we consider

   i) Magnitude, $|F(u, v)| := \sqrt{F_{\mathrm{r}}(u, v)^2 + F_{\mathrm{i}}(u, v)^2}$, and

   ii) Phase Angles, $\theta(u, v) := \tan^{-1}\left(\dfrac{F_{\mathrm{i}}(u, v)}{F_{\mathrm{r}}(u, v)} \right)$

Remark 3.3 - *Expressing $F(u, v)$ in Polar Cordinates*

$$F(u, v) = |F(u, v)| e^{i\theta(u,v)}$$

Remark 3.4 - *Plotting Magnitude, $|F(u, v)|$*
On the left we have a gray scale image & on the right we have the magnitude of a fourier transform on this image. On the right hand image the y-axis is $u \in [-\pi, \pi)$ and the x-axis is $v \in [-\pi, \pi)$. We see lots of straight lines since a linear transformation on $F(u, v) = F(au, av) \; \forall \; a \in \mathbb{R}$. Each line can be interpreted as the frequency of intensity for lines in the left hand image which are parallel to it.
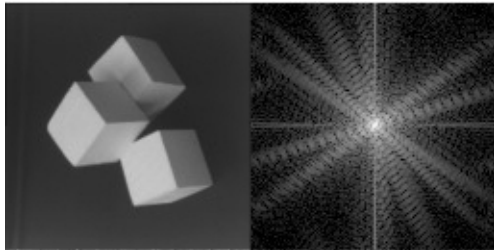
Theorem 3.2 - *Convolution Theorem*
Let f be an image, g be a kernel, F be the result of a fourier transform on f and G be a kernel. Then

$$h = f * g \iff H = FG$$

Proof 3.1 - *Convolution Theorem*

$$
\begin{aligned}
h(x) &= f(x) * g(x) \\
&= \sum_y f(x-y)g(y) \\
H(u) &= \sum_x \left(\sum_y f(x-y)g(y) \right) e^{iux} \\
&= \sum_x g(y) \sum_x f(x-y) e^{iux} \\
&= \sum_y g(y) \left(F(u) e^{iux} \right) \\
&= \sum_y g(y) e^{iuy} F(u) \\
&= G(u) \cdot F(u) \\
&= F(u) \cdot G(u)
\end{aligned}
$$

Definition 3.4 - *Butterworth's Low Pass Filter*
Butterworth's Low Pass Filter is a *Signal Processing Filter* designed to have a frequency response which is as flat as possible. It appears to soften an image

$$
H(u,v) = \frac{1}{1 + \left(\dfrac{r(u,v)}{r_0} \right)^{2n}} \text{ of order } n
$$

Definition 3.5 - *Butterworth's High Pass Filter*
Butterworth's High Pass Filter is a *Signal Processing Filter* designed to have a frequency response which is as flat as possible. It appears to sharpen an image

$$
H(u,v) = \frac{1}{1 + \left(\dfrac{r_0}{r(u,v)} \right)^{2n}} \text{ of order } n
$$

4   Edges & Shapes

4.1   Edge Detection

Remark 4.1 - *Edges are Useful*
Edges are one of the best features to identify an object with. This *Edge-Detection* is a useful thing to be able to do.
Definition 4.1 - *Edge*
An *Edge* is a sharp change in image brightness. Edges are produced by object boundaries, patterns & shadows, this can lead to us finding *Nusiance Edges* which do not help achieve our goal.

Example 4.1 - *Edge Detection*
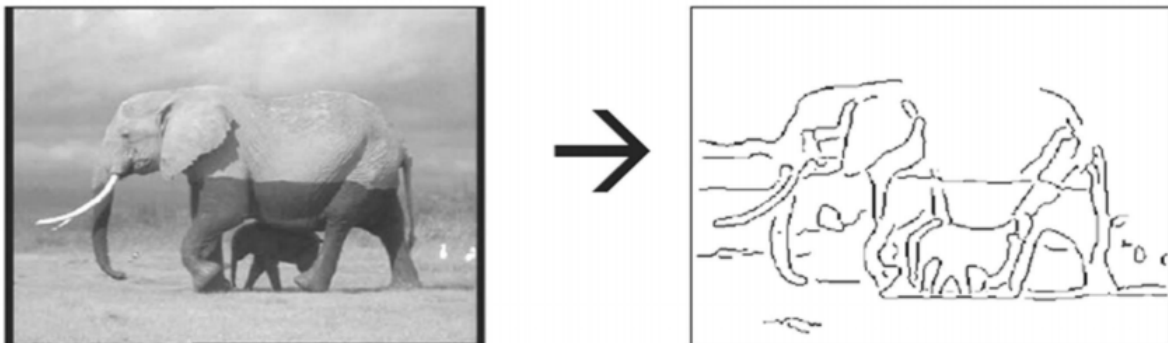
Remark 4.2 - *Uses of Edges*
Edges are used for

   i) Segmentation - Finding object boundaries;

   ii) Recognition - Extracting patterns;

   iii) Motion Analysis - Finding reliable tracking regions.

Remark 4.3 - *Edge Detection Strategy*
In order to find edges we want to determine the *rate of change* in a pixel's neighbourhood
Definition 4.2 - *Image Gradient*
Image Gradient is a vector which gives the direction of greatest change in intensity at a specific pixel. For a pixel at (x, y) with $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ mapping to the intensity the *Image Gradient* is

$$\nabla f(x, y) := \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

N.B. Generally denoted by Ψ.

Definition 4.3 - *Angle of Gradient*

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

Definition 4.4 - *Edge Direction*
Edge Direction for a specific pixel is the direction of an edge, it is perprendicular/orthogonal to the *Image Gradient.*
N.B. Generally denoted by Φ.

$$\Phi := \Psi - \frac{\pi}{2} = \tan^{-1}\left(\frac{\partial f/\partial y}{\partial f/\partial x}\right) - \frac{\pi}{2}$$

Definition 4.5 - *Image Gradient Magnitude*
*Magnitude* measures the magnitude of the growth.

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

Proposition 4.1 - *Estimating ∇f*
We can define $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ to be matrices, rather than typical derivatives, in order to produce estimates for the *Image Gradient* which can be used to analyse an image by convolution. We can then combine the results in order to analyse *Image Gradient Angle* and *Magnitude*. Consider

$$\frac{\partial f}{\partial x} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \text{ and } \frac{\partial f}{\partial y} = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$
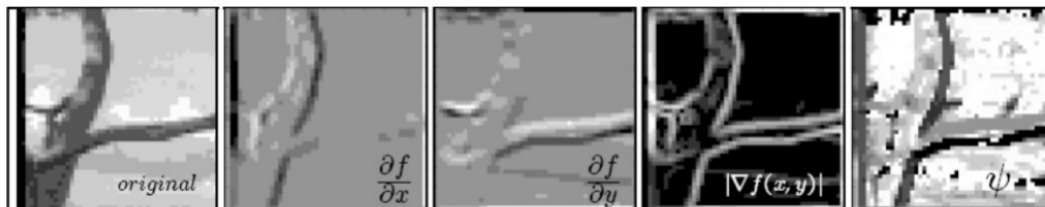
Definition 4.6 - *Prewitt Operator*
*Prewitt Operator*s are a set of 3×3 kernels used for estimating the *Image Gradient*. There are 8 *Prewitt Operators*, each in a different direction

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{pmatrix} \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{pmatrix}$$

Remark 4.4 - *Alternative estimates of* ∇f
Instead of weighting all pixels in a given direction equally, we may want to weight central ones more heavily. This is since the central pixels are geometrically closer to the pixel which is being acted upon & such should better describe it.

$$\frac{\partial f}{\partial x} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \frac{\partial f}{\partial y} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

N.B. This is called the *Sobel Filter*.

4.2   Shapes

Definition 4.7 - *Hough Transform*
A *Hough Transform* is used to find lines which best explain the set of edge points given to it.
Hough Transforms are centred around the premise that lines can be described by $\rho = x\cos\theta + y\sin\theta$ where either (x, y) is fixed, or (ρ, θ) is fixed.
Suppose we are given an edge point (x_0, y_0), we are now dealing with the case where (ρ, θ) are variable. Plot all the combinations of (ρ, θ) than produce lines which pass through (x_0, y_0) produces a sinusoidal wave.
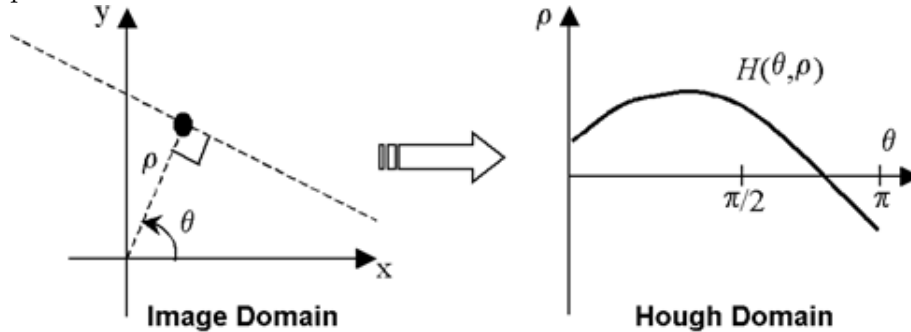
Proposition 4.2 - *Line Detection Algorithm*
Below is an algorithm which uses the *Hough Transform* on a set of edge points in order to detect lines in an image

   i) Make available an $n = 2$ dimensional array, $H(\rho, \theta)$, to be used for the parameter space;

   ii) Find the *Gradient Image*, $G(x, y)$;

   iii) For any pixel (x_0, y_0) where $|G(x_0, y_0)| > T_s$ (some threshold value) increment all values of $H(\rho, \theta)$ where (ρ, θ) satisfy $\rho = x_0 \cos\theta_0 + y_0 \sin\theta_0$.

$$\forall\, \rho, \theta \text{ where } \rho = x_0 \cos\theta_0 + y_0 \sin\theta_0 \text{ do } H(\rho, \theta) + = 1$$

   iv) Any (ρ, θ) where $H(\rho, \theta) >= T_h$ (another threshold value) represent a straight line which has been detected in the image.

Proposition 4.3 - *Circle Detection Algorithm*

   i) Make available an $n = 3$ dimensional array, $H(x, y, r)$, to be used for the parameter space;

ii) Find the *Gradient Image*, $G(x, y)$;

iii) For any pixel (x_0, y_0) where $|G(x_0, y_0)| > T_s$ increment all $H(x, y, r)$ which satisfy

$$\forall \ r \text{ where } x_0 = x + r \cos \Phi \text{ and } y_0 + y + r \sin \Phi$$

iv) Any (x, y, r) where $H(x, y, r) > T_h$ represents a circle with radius r and centre (x_0, y_0)

Definition 4.8 - *General Hough Transform*
A *Genereal Hough Transform* is used to describe general shapes.

i) Find the *Gradient Image*, $G(x, y)$.

ii) Define a set of points $\phi_i := \frac{pi}{i}$ for $i \in [1, k]$ and create a table using $\phi_1, \ldots, \phi_k$ as indexes.

iii) Define a reference point (x_c, y_c) to act as a *centre of mass*.

iv) For any given edge point (x_0, y_0) find

$$r = \sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2}, \ \beta = \tan^{-1}\left(\frac{y_0 - y_c}{x_0 - x_c}\right) \text{ and } \Phi_{(x_0, y_0)} \text{ in } G(x, y)$$

v) Round $\Phi_{(x_0, y_0)}$ to the nearest ϕ_i and insert (r, β) into the table at ϕ_i

Proposition 4.4 - *General Shape Detection Algorithm*

i) Prepare 2 dimensional array $H(x_c, y_c)$ for the parameter space (Set of all possible centre of masses).

ii) $\forall \ (x_0, y_0)$ where $|G(x_0, y_0)| > T_s$ find the table entry, ϕ_i, closest to $\Phi_{(x_0, y_0)}$.

iii) $\forall \ (r_j, \beta_j)$ in the table entry find

$$x_c = x + r \cos \beta \text{ and } y_c = y + r \sin \beta$$

Increment $H(x_c, y_c)$.

iv) All (x_c, y_c) where $H(x_c, y_c) > T_h$ represents the locations of centre of masses for occurances of the shap, in the image.

Remark 4.5 - *Issues with **Proposition 4.4***
The algorithm defined in **Proposition 4.4** only detects shapes of the same *scale* and *orientation* as the original image. To counter this we introduce two new variables to the parameter space: S, a scaling factor; and, θ, an orientation factor.

i) Prepare 4 dimensional array $H(x_c, y_c, S, \theta)$ for the parameter space (Set of all possible centre of masses, scales & orientations).

ii) $\forall \ (x_0, y_0)$ where $|G(x_0, y_0)| > T_s$ find the table entry, ϕ_i, closest to $\Phi_{(x_0, y_0)}$.

iii) $\forall \ (r_j, \beta_j)$ in the table entry find

$$x_c = x + rS \cos(\beta + \theta) \text{ and } y_c = y + rS \sin(\beta + \theta)$$

Increment $H(x_c, y_c, S, \theta)$.

iv) All (x_c, y_c) where $H(x_c, y_c, S, \theta) > T_h$ represents the locations of centre of masses, scaling factors & orientations for occurances of the shape in the image.

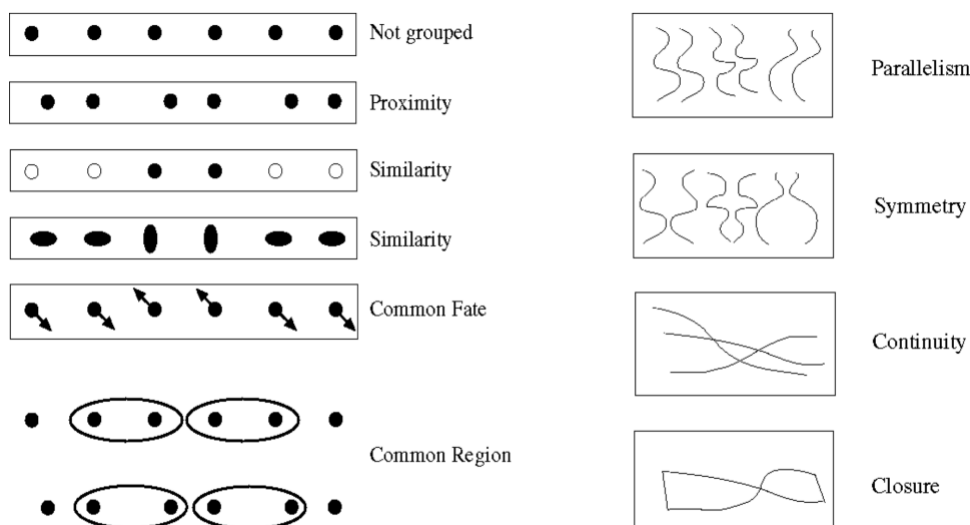# 5   Image Segmentation

Definition 5.1 - *Image Segmentation*
Image Segmentation is the process of partitioning the pixels in an image into homogenous regions. This may wrt colour, texture, object.

Remark 5.1 - *Usefulness of Image Segmentation*
Image Segmentation is useful since it simplifies an image from millions of pixels into a few regions making computation much easier.

Definition 5.2 - *Gestalt Rules for Grouping & Segmentation*
It is useful to consider the different ways in which we can group objects.

Remark 5.2 - *Perfect Image Segmentation is hard*
It is hard to achieve perfect *Image Segmentation* since pixels may straddle the boundary of objects & thus not truely belong to any object. Noise, non-uniform illumination, occlusions etc. will also cause problems.

Definition 5.3 - *Over-Segmentation*
Over-Segmentation is when an image is segmented into too many regions & loses the efficienies gained by the simplication of *Image Segmentation*.
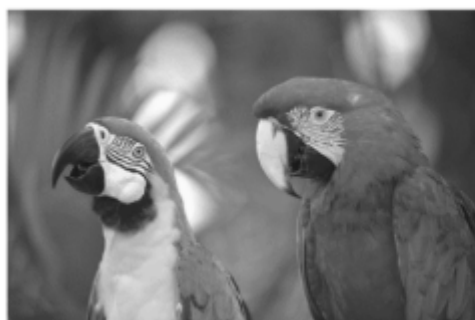
Example 5.1 - *Over-Segmentation*

Definition 5.4 - *Under-Segmentation*
Uner-Segmentation is when an image is segmented into too few regions & too much information is lost.

Example 5.2 - *Under-Segmentation*

Proposition 5.1 - *Segmentation Technique Types*
There are several groups of techniques that can be used for *Image Segmentation*

   i) Thresholding. Categorise pixels wrt intensity.

   ii) Edge-Based. Region boundaries are produced using an edge map.

   iii) Region-Based. Regions are grown from seed pixels or using split-merge techniques.

   iv) Clustering & Statistical. Global partioning, oftern based around histograms. (*e.g.* K-means).

   v) Topographic (OUT OF SCOPE). Stepwise simplications that take spatially wider image configurations into account.

Proposition 5.2 - *Thresholding Image Segmentation*
Thresholding is a good technique when trying to distinguish dark objects(background) from bright objects (foreground). One possible algorithm is

   i) Choose a threshold value T.

   ii) For each pixel:

      (a) If the brighness is less than T map to 0 (black).
      (b) else, map to 255 (white).

Remark 5.3 - *Choosing Threshold Value*
Choosing a value for *Thresholding* is important since if it is too high then the background pixels are classified as the foreground. And visa-versa for too low.

Proposition 5.3 - *Choosing Threshold Value*
There are several techinques used for choosing *Threshold Value*. By consider the histogram of an image's intensity we can produce regions for classification.

   i) Make an initial estimate for threshold T.

   ii) Segment the image using T: Let G_1 be the pixels with intensity $> T$; and G_2 be the pixels with intensity $< T$.

   iii) COmpute the average intensity values m_1 & m_2 for G_1 & G_2 respectively.

   iv) Compute a new threshold value $T = \frac{1}{2}(m_1 + m_2)$.

   v) Repeat **ii)-iv)** until convergence.

Proposition 5.4 - *Edge-Based Segmentation*
When performing *Edge-Based Image Segmentation* we want to detect strong edges that are relevant to the objet we wish to detect. There are several techniques for assessing the strength of an edge, but generally we wish to remove edges that are isolated of irrelevant to an object. One valid technique is edge relaxation where we change the strength of an edge depending upon its 6 neighbour edges.
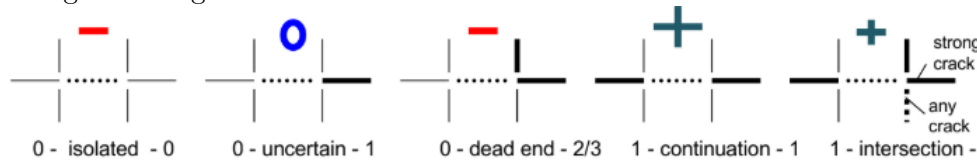
Proposition 5.5 - *Region Growing-Seed Pixel*
A function is required to analyse the homogeneity of two pixels $(s, p) \rightarrow \{0, 1\}$.

   i) Choose an initial seed pixel;

   ii) Consider the neighbouring pixels:

      (a) If it is close to seed pixel, add to region & to queue to be analysed.
      (b) Else, do nothing with it.

   iii) Repeat **ii)** until there are no pixels left in the queue to be analysed.

N.B. You may wish to add a final step to remove small regions.

Proposition 5.6 - *Split & Merge Segmentation*
Let H be a function to analyse the homogeneity of a region $(R) \rightarrow \{0, 1\}$

   i) Let R_0 represent the entire image.

   ii) If $H(R_i) = 0$ (*i.e.* is inhomogenous) then split it into four regions.

   iii) Repeat **ii)** until all regions are homogenous.

   iv) Merge all subregions that satisfy $H(R_i \cup R_j) = 1$ (*i.e.* are homogeneous)

Proposition 5.7 - *Clustering Segmentation*
If we map the RGB values of an image into the 3D real space we can we clustering algorithms to produce k clusters. We then map back to the pixel space using each cluster as a segment.
N.B. Look at k-means clustering.

6   Classical Object Detection

Remark 6.1 - *Classical Object Detection*
In this course we only conver *Classical Object Detection*. This technique came to prominence with the 2001 paper *First Real-Time Detection Method: Viola & Jones' - 2001* and was the industry standard for 10+ years. Now there are methods which use deep learning techniques.

Definition 6.1 - *Object Detection*
Object Detection bridges the semantic gap between [ixel values & meaningful objects by grouping pixels and classifying. This is a difficult problem.

Remark 6.2 - *Failings of Shape Detection for Object Detection*
Typically using classical *Shape Detection & Segmentation* does not work for real world applications of *Object Detection* due

i) High Intra-Class & Low Inter-Class variance;

ii) Classes are rarely well defined;

iii) Variations in illumination, scale, pose, deformation, occlusion etc.

Remark 6.3 - *Shift-Scale Invariance*
We require *Object Detection* algorithms to be shift & scale invariant so it can detect objects of any size at any point in the image.

Definition 6.2 - *Sliding Window Detectors*
Sliding Window Detectors ensure shift & scale invariance by setting a minimum size to check (say 20×20px) and shifts the region it checks by 1px right-to-left, top-to-bottom until the whole is checked. They then increase the size of the region it checks (say 10%) and repeats the shifting. It does this until the size of the region it checks exceeds the size of the image.
*N.B.* This technique does not work in real-time due to its high computation complexity.

Definition 6.3 - *Haar-Like Features*
Haar-Like Features are features of digital images used in *Object Detection*. They define a set of adjacent rectangles and assigns them to either be *light* or *dark* regions. It then returns a value, the average gray value of the light region less the average gray value of the dark region. This value is then compared to a threshold value to determine whether the feature occurs in the image.

Proposition 6.1 - *Haar-Like Feature Types*
There are a finite set of *Haar-Like Features Types*
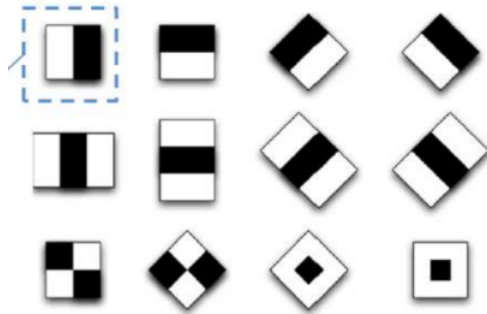
Definition 6.4 - *Image Integral*
The *Integral* of an image I is the image J where $J(x,y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i,j)$. *i.e.* Each pixel value is the sum of all pixels between it an the origin, inclusive.

Proposition 6.2 - *Calculating Image Integral*
Let I be an image then we can calculate the *Integral* of J in *Linear Time* using the following formulae
$$A(x, -1) = 0$$
$$A(x, y) = A(x, y - 1) + I(x, y)$$
$$J(-1, y) = 0$$
$$J(x, y) = J(x - 1, y) + A(x, y)$$

Theorem 6.1 - *Integration Rule of Convolution*

Proposition 6.3 - *Calculating the Average Pixel Value of a Large Block*
Calculating the average pixel value of a large block can be very slow if the region is very large. Here is a technique that can do it by only quering 4 values.

i) Create a matrix which is the same size as the image.

ii) Encode the matrix st its values are 1 in the region we want to check & 0 everywhere else.

iii) Double Differentiate this matrix to produce I''

    (a) First, horizontally by applying $\begin{pmatrix} 0 & -1 & 1 \end{pmatrix}$ to every pixel.

    (b) Then, vertically by applying $\begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$ to every pixel.

    This leaves us with four non-zeros values (two 1s and two -1s).7

iv) Calculate the image integral J.

v) Return $\sum_i \sum_j I''(i,j) \times J(i,j)$
    N.B. This can be simplied to just be the four non-zero terms in I''.

Proposition 6.4 - *Training Data*
Two categories of *Trainging Data* should be provided: *Positive Samples*, images of the object you wish to classify' and, *Negative Samples*, images which do not contain the object. These should be labled (x_i, y_i) with $y_i = 0, 1$ for *Negative* and*Positive Samples* respectively.

Definition 6.5 - *Adaboost*
Adaboost (Short for *Adaptive Boosting*) is a machine learning algorithm.
Let $(x_1, y_1), \ldots, (x_n, y_n)$ with $y_i = 0, 1$ for negative & positive samples respectively

i) Initalise weights $w_{1,i} = \begin{cases} \frac{1}{2m}, & \text{if } y_i = 0 \\ \frac{1}{2l}, & \text{if } y_i = 1 \end{cases}$ where m is the number of negative samples & l is
    the number of positive samples.

ii) For $t = 1, \ldots, T$ where T is the number of features you wish to find

    (a) Normalise the weights st w_t is a probability distribution

$$w_{t,i} := \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

    (b) For each feature, j, train a classifier h_j which is restricted to using a single feature (Haar-Like Feature). Calculate the error wrt w_t

$$e_j = \sum_i w_i |h_j(x_i) - y_i|$$

    (c) Choose the classifier, h_t, with the lowest error e_t.
    (d) Update the weights

$$w_{t+1,i} = w_{t,i} \left(\frac{e_t}{1 - e_t} \right)^{1 - e_i}$$

    where $e_i = 0$ if sample x_i is classified correctly and $e_i = 1$ otherwise

iii) The final strong classifier is

$$h(x) = \begin{cases} 1 & \sum_{i=1}^{T} \log \left(\frac{e_t}{1 - e_t} \right) h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \log \left(\frac{1 - e_t}{e_t} \right) \\ 0 & \text{otherwise} \end{cases}$$

7   Motion

Definition 7.1 - *Video Sequences*
A video can be considered as a series of matrices, each representing an image at a different point in time. The time difference between consecutive matrices will be constant, Δt seconds.
Define $I(x, y, t)$ to be the intensity of the pixel at position (x, y) at time t.

Definition 7.2 - *Frame Rate*
Frame Rate is the number of images per second in a video. Typically measured in FPS (*Frames Per Second*), this is equivalent to Hz.

$$\text{Frame Rate} = \frac{1}{\Delta T}$$

N.B. Typical frame rates are $24fps$, $30fps$ & $60fps$.

Definition 7.3 - *2D Tracking*
2D Tracking is tracking how an object moves in a two dimensional plane. The object may really be moving a three dimensional plane, but we only care about its movement in a specified 2D plane.
This means the object has 3 attributes: x position, y position & orientation. Scale can be considered too.

Example 7.1 - *2D Tracking*
In the below image the path the pen has taken in the plane parallel to the camera has been tracked. The pen is actually moving in 3-dimensions, but we are ignoring depth in this example.
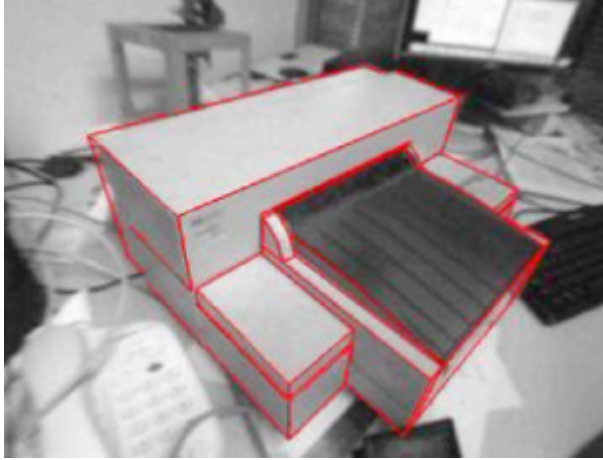

Definition 7.4 - *3D Tracking*
3D Tracking is used to track an object's movement in a three dimensional space. This means an object has six factors: x,y,z positions and rotation in the x,y,z directions. Scale can be considered too.

Example 7.2 - *3D Tracking*
In the below image a printer is tracker so that a skeleton of it can be mapped to it.

**Definition 7.5 -** *Simultaneous Localisation and Mapping*
Simultaneous Localisation and Mapping, SLAM, is the process of constructing and updating a map of an unknown enviroment while tracking the camera's position within it.
N.B. This can be made easier by using a depth sensor.

Definition 7.6 - *Perspective Projection Equations*
Suppose a 3D point $\mathbf{P} = (X, Y, Z)$, which sits on the surface of an object, is projected to a point in two dimensions, $\mathbf{p} = (x, y, f)$ (f is the distance of the 2D plane from the camera). Then, by similar triangles

$$x = \frac{fX}{Z} \text{ and } y = \frac{fY}{Z}$$
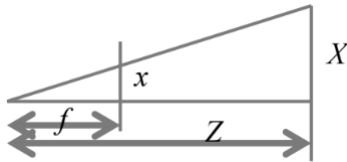
Definition 7.7 - *Rotation Matrices*
A *Rotation Matrix* can be applied to a position, by matrix-multiplication, to perform a rotation on that position. A *Rotation Matrix*, R can be broken down into *Rotation Matrices* which represent the rotation about each axis. In three-dimensions this means $R = R_X R_Y R_Z$.
N.B. Rotation Matrices are $n \times n$ in n-dimensional space.

Remark 7.1 - *3D Rotation Matrices*

$$R_X(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \quad R_Y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \quad R_Z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Proposition 7.1 - *Rotation Matrices for small θ*
When θ is small we can approximate $\sin\theta \approx \theta$ and $\cos\theta \approx 1$. Thus

$$R_X(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -\theta \\ 0 & \theta & 1 \end{pmatrix} \quad R_Y(\theta) = \begin{pmatrix} 1 & 0 & \theta \\ 0 & 1 & 0 \\ -\theta & 0 & 1 \end{pmatrix} \quad R_Z(\theta) = \begin{pmatrix} 1 & -\theta & 0 \\ \theta & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Definition 7.8 - *Rigid Object*
A *Rigid Object* is one which does not change in shape, only scale & rotation.

Proposition 7.2 - *3D Rigid Motion*
Consider a *Rigid Object* & a point on that object $\mathbf{P}$.

If the object is moved st point $\mathbf{P}$ is now at position $\mathbf{P}'$ then we can describe $\mathbf{P}'$ in terms of $\mathbf{P}$ by defining a translation matrix T and a rotation matrix R.

$$\mathbf{P}' = R\mathbf{P} + T$$

N.B. $\mathbf{P}$, $\mathbf{P}'$ & T are 3×1 matrices and R is a 3×3 matrix.

Proposition 7.3 - *3D Motion Field*
Consider a *Rigid Object* & a point on that object $\mathbf{P}$.
Let V be the vector which represents the translation from $\mathbf{P}$ to $\mathbf{P}'$. We have

$$V = \lim_{\Delta t \to 0}[\mathbf{P}' - \mathbf{P}] = \lim_{\Delta t \to 0}[(R - I)\mathbf{P} + T]$$

We note that for small θ_X, θ_Y & θ_Z

$$R = \begin{pmatrix} 1 & -\theta_Z & \theta_Y \\ \theta_Z & 1 & -\theta_X \\ -\theta_Y & \theta_X & 1 \end{pmatrix}$$

Thus

$$\begin{array}{rcccl} \frac{dX}{dt} & = & V_X & = & \theta_Y Z - \theta_Z Y + T_X \\ \frac{dY}{dt} & = & V_Y & = & \theta_Z X - \theta_X Z + T_Z \\ \frac{dZ}{dt} & = & V_Z & = & \theta_X Y - \theta_Y X + T_Z \end{array}$$

N.B. Here $(\theta_X, \theta_Y, \theta_X)$ is called the *Angular Velocity* and (T_X, T_Y, T_Z) is called the *Rectilinear Velocity* since we are taking the limit as Δt tends to 0.

Proposition 7.4 - *2D Motion Field Equations*
Consider a point $\mathbf{p} = (x, y, f)$ which represents a 2D projection of a point $\mathbf{P} = (X, Y, Z)$.
Then

$$\begin{array}{rcl} v_x & = & \frac{d}{dt}x \\ & = & \frac{d}{dt}\left(\frac{fX}{Z}\right) \\ & = & f\frac{V_X Z - XV_Z}{Z^2} \text{ by quotient rule} \end{array}$$

Substituting V_X, V_Y, V_Z for the definitions in **Proposition 7.3** we get

$$v_x = \frac{fT_X - xT_Z}{Z} + f\theta_Y - \theta_Z y - \frac{\theta_X xy - \theta_Y x^2}{f} \text{ and } v_y = \frac{fT_Y - yT_Z}{Z} - f\theta_X + \theta_Z x + \frac{\theta_Y xy - \theta_X y^2}{f}$$

We can recognise two components of these equations: Translation, terms involving T; and Rotation, terms involving θ.

7.1   Motion Estimation

Definition 7.9 - *Motion Estimation*
Motion Estimation is the process of using spatial & temporal variation in pixel values to detet amount of movement in an image region.

Definition 7.10 - *Optical Flow*
Optical Flow is the movement of pixel values between frames. We assume that pixels follow a trajectory through frames. *i.e.* There is a sequence $(x(t), y(t))$ that shows how a pixel moves from frame to frame.
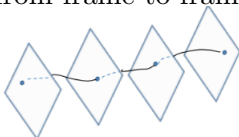
N.B. Also known as *Apparent Motion.*

Remark 7.2 - *Optical Flow & True Motion*
In some cases the *Optical Flow* in an image is not equivalent to the *True Motion* of the object. Consider a diagonal rod moving across the screen, it is generally very hard to tell if it is moving horizontally or vertically or any other direction.
N.B. This is sometimes the result of the *Aperture Problem.*

Definition 7.11 - *Optical Flow Equation*
Let $I(x, y, t)$ be the grayscale value of pixel (x, y) at time t. If we wish to follow a particular pixel between frames we find that $I(x, y, t)$ is fixed & (x, y) depend on t. Further

$$\frac{d}{dt}I(x, y, t) = 0$$

By the chain rule we have

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \implies I_x v_x + I_y v_y + I_t = 0$$

N.B. Gradient$= (I_x, I_y, I_t)$ and Motion$=(v_x, v_y)$.

Remark 7.3 - *Estimating the Optical Flow Equation*
The *Optical Flow Equation* has two unknowns v_x & v_y. This means we cannot solve it for a single pixel, as it would be underconstrained. Hence we have to *estimate* its value by using multiple pixels.
Often a 2×2 square of pixels is used to estimate the *Normal Flow* of the central point.
N.B. The number of pixels used to *estimate* depends on the complexity of v_x & v_y. If v_x & v_y are linear in x & y with 3 parameters (*e.g.* $v_x = ax + by + c$) then four pixels is sufficient.

Proposition 7.5 - *Constant Velocity Model*
Suppose we have a region R of the image (often taken as the whole image) we can assume it has constant velocity (v_x, v_y) accross it. Then our problem is to find v_x, v_y which minimise

$$\varepsilon(v_x, v_y) = \sum_R (I_x v_x + I_y v_y + I_t)^2$$

N.B. This is a minimisation problem since, ideally, $I_x v_x + I_y v_y + I_t = 0$.

Theorem 7.1 - *Lucas-Kanade Algorithm*
The *Lucas-Kanade Algorithm* aims to solve the problem in **Proposition 7.5**.
Since it is a minimisation problem we take the derivatives and find

$$\frac{\partial \varepsilon}{\partial v_x} = 2\sum_R I_x(I_x v_x + I_y v_y + I_t) \quad \text{and} \quad \frac{\partial \varepsilon}{\partial v_y} = 2\sum_R I_y(I_x v_x + I_y v_y + I_t)$$

Equating these to zero and solving we find

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = A^{-1}b \text{ where } A := \sum_R \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \text{ and } b = -\sum_R \begin{pmatrix} I_t I_x \\ I_t I_y \end{pmatrix}$$

Remark 7.4 - *Motion for Object Detection*
Motion Estimates can be a good cue for object detection (and segmentation) as it implies depth in the image. However it requires an object to have an approximately constant velocity otherwise it will be partitioned (consider trying to detect the body of a person who is waving their

arm).

Definition 7.12 - *Layered Representation*
If the camera is moving with a constant motion we can infer depth from the image by identifying motion-homogeneous regions. We can use this information to produce a *Layered Representation* of the image, where each layer is a *depth-homogeneous* region.

Definition 7.13 - *Layered Video*
Every frame of an image can be split into three maps

1. Colour Map - The colour of each pixel in the layer.

2. Alpha Map - The pixels which are in the layer.

3. Velocity Map - The *Motion Estimate* for pixels in the layer.

Proposition 7.6 - *Motion Segmentation Process*

1. Perform *Block Motion Fitting*.

    (a) Split the image into blocks of equal area.

    (b) Estimate the motion in each block.

2. Detect the dominant motions using clustering algorithms and returning the centroids of each cluster.

3. Perform *Motion Segmentation* by assigning pixels to the dominant motion which is closest to theirs

4. Produce Final Parameteric Motion Fitting by updating our motion estimates using our assigned pixel motions.

Proposition 7.7 - *Region Linking & Alignment*
Once we have used the process in **Proposition 7.6** we can pick a particular region & align it across every frame. By taking the median values of every pixel in this aligned form we can produce the maps mentioned in **Definition 7.13**

0   Reference

0.1   Definitions

Definition 0.1 - *Connectivity*
When analysing a pixel there are several ways to consider its adjacent neighbourhood.

- 4-Connectivity. The pixels above, below, left & right.

- 8-Connectivity. Above, below, left, right & single step diagonals.

Definition 0.2 - *Kernel*
A *Kernel* is a small matrix used in convolution. Typically 3×3 or 5×5. *Kernels* can be defined for blurring, sharpening, embossing, edge detection & more *N.B.* This definition only applies to image processing & is different from the definition in linear algebra.