

IVR Condensed Summary Notes For Quick In-Exam Strategic Fact Deployment

Maksymilian Mozolewski

December 6, 2020

Contents

1	Introduction to Vision	2
2	Image Basics	4
3	Image Segmentation	10
4	Description of Segments	15
5	Object recognition	20

Introduction to Vision

Computer vision

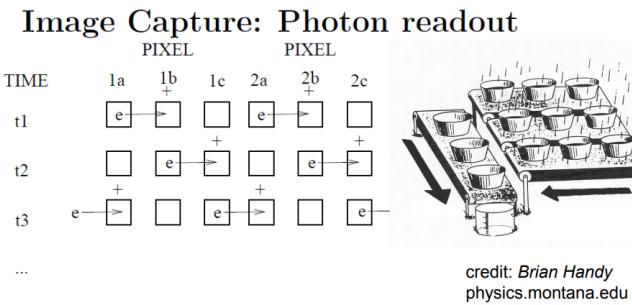
Processing data from any modality which uses the electromagnetic spectrum and produces an image

Image

Way of representing data in a picture-like format, with a direct correspondence to the scene being imaged

CCD Camera

Charged couple device, light falls on an array of MOS capacitors (which are rectangular and not square). The capacitors form a shift register and output either a line at a time or the whole array at one time (line vs frame transfer)



these "buckets" can overflow, resulting in over-saturation of the image

Frame grabber

Device which converts analog image signals to digital image signals. Essentially puts a discrete value on each pixel signal. 24bit color is usually required for robotics

Visual Erosion

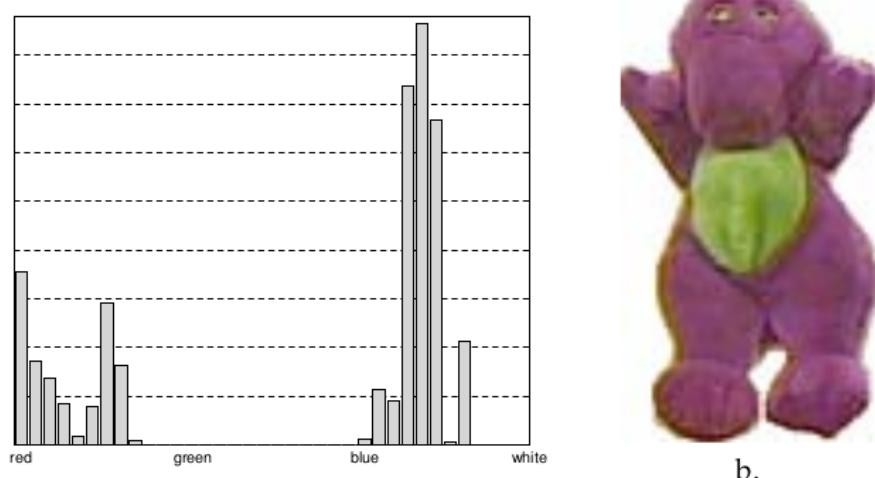
RGB is a function of the sensitivity of the sensor to reflected light of each color. The sum of those intensities may vary wildly from frame to frame depending on the distance of the object due to intensity of the reflected light. The object appears to "erode" with changes in lighting. CCD Cameras are also notoriously insensitive to red, meaning that one of the three color planes is not as helpful in distinguishing colors. HSI and SCT colour spaces aim to reduce visual erosion since the Hue - the main wavelength measured (**perceptually meaningful dimensions**) will not change with the object's relative position, only its saturation and intensity will! Equipment to capture HSI images is expensive, and conversions between colour spaces sometimes fail.

Region Segmentation

Finding groups of pixels related to each other via color, within a certain threshold and identifying the centroids of those groups. Requires high contrast between the **foreground** (object of interest) and the **background** to work well.

Color histogramming

a type of histogram (bar chart basically), the user specifies range of values for each bar, (bucket) the size of the bar is the number of data points falling within the bar's "range". These ranges could be set to capture different values of either R,G,B color intensities.



Such histograms can be **subtracted bucket-wise** from each other as a form of distance measure to compare image stimuli.

Stereopsis

The method of triangulating depth data from 2 POV's

Stereo camera pairs

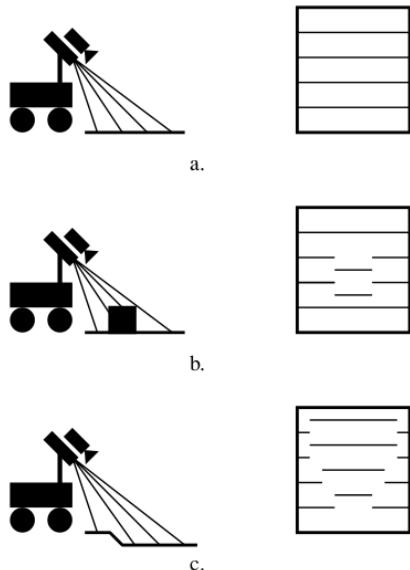
Usage of two cameras to extract range data by finding the same point on the images received from two (most likely parallel) cameras, and then finding the depth information using the geometry of the cameras. It can be hard to find the same point on two pictures (**correspondence problem**), the method of picking a spot of interest is called an **interest operator**. Cameras can be mounted in parallel to produce **rectified images** (the distance between the two cameras is then known as the **disparity**). This can save computation time since the point of interest will appear in the same line of the image on both cameras (**epipolar lines**)

Optic flow

Information to do with: Shadow cues, texture, expected size of objects

Light stripping

Method of projecting a pattern of light onto a surface of interest and observing the distortion to the pattern to visualise the surface and/or distance information. Does not work that well in natural conditions due to noise.



Laser ranging

like radar but using light (**lidar**), scanning components are expensive, a planar laser range finder is a cheaper alternative. Produces an intensity and range map.

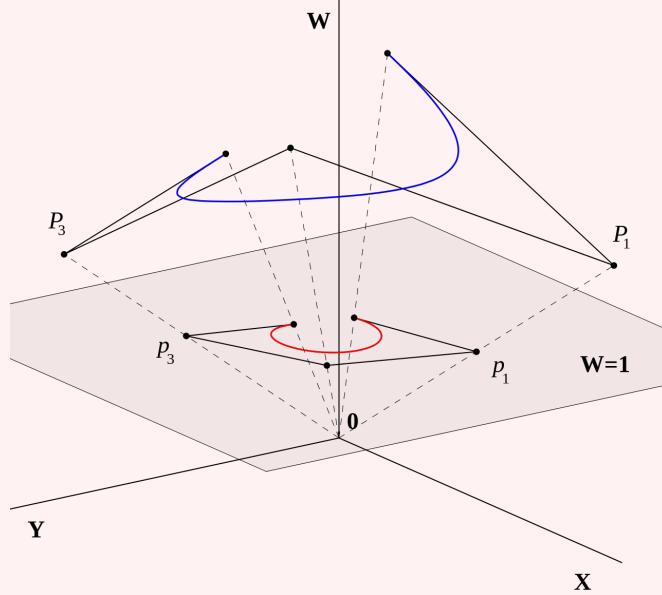
Range segmentation

Segmenting the image based on range data, can be used to determine the geometry of surfaces

Image Basics

Homogenous coordinates

Homogenous (aka similar) coordinates are coordinates in space with one more dimension than in the corresponding cartesian space, in this space we can express linear translations as linear matrix transformations! Every point in the cartesian space becomes a line in the homogenous space!



Conversion to homogenous coordinates:

$$\begin{bmatrix} x \\ y \\ \vdots \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ \vdots \\ 1 \end{bmatrix} \quad (1)$$

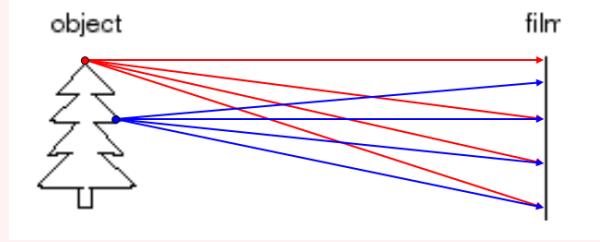
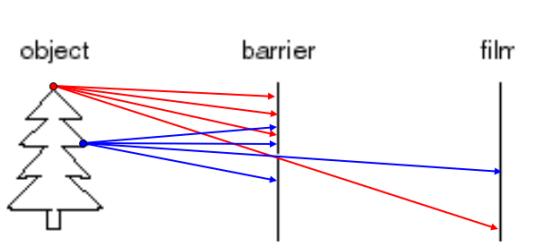
Conversion from homogenous coordinates

$$\begin{bmatrix} x \\ y \\ \vdots \\ w \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ \vdots \\ 1 \end{bmatrix} \quad w \neq 0 \quad (2)$$

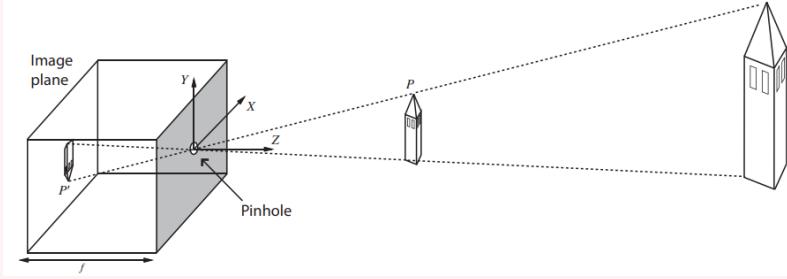
Notice how a point in homogenous space can be multiplied by any constant, and yet when it is converted back to normal space, it becomes the same point. **The ratio** between the components defines the line in homogenous space.

Pinhole camera

Capturing on a simple plane does not work because multiple rays from the same point in the scene travel to multiple parts of the film. We want the film to capture a single "ray" per point of interest



A camera setup using a tiny hole to filter and hence focus the light onto a single clear image.



Using similar triangles, the point $P:(X, Y, Z)$ maps to point P' on the 2d surface of the image plane, at a distance f (**focal length**) from the pinhole as follows:

$$x = \frac{-fX}{Z}, y = \frac{-fY}{Z}, z = f \quad (3)$$

This projection of scene point to camera point can be expressed as a linear matrix transformation in homogenous space:

$$P_h = \begin{bmatrix} X \\ Y \\ -Z/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (4)$$

To retrieve the projected point in cartesian space we simply divide by the third coordinate and discard it.

$$P_c = \begin{bmatrix} X/(-Z/f) \\ Y/(-Z/f) \end{bmatrix} = \begin{bmatrix} -fX/Z \\ -fY/Z \end{bmatrix} \quad (5)$$

Which is identical to the projection above.

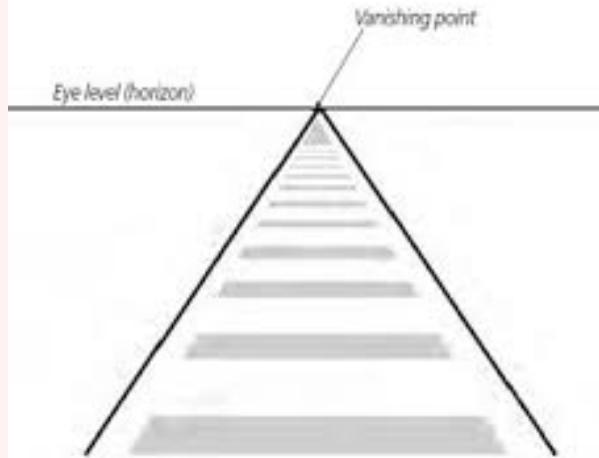
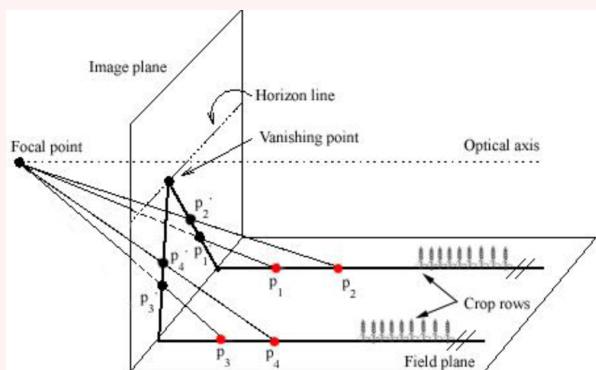
This projection, preserves straight lines (**colinearity**) and their intersections, but loses information about angles and lengths (due to multiple points in 3D possibly mapping to the same point in 2D)

Lines directly passing through the focal point are projected as points.

Planes are preserved but those passing through the focal point are projected as lines.

Vanishing point

Any two parallel lines will converge to a certain point on the image as long as their directions are the same



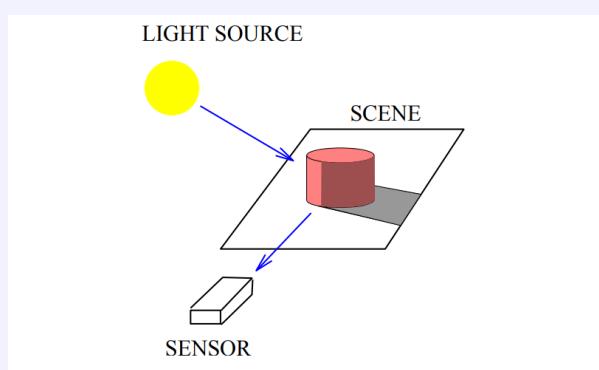
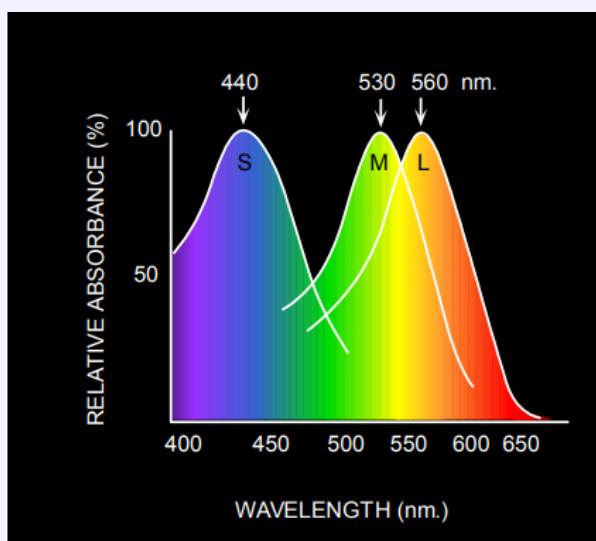
Detector response curve

The curve showing which frequencies of light a detector perceives the most and which will dominate the actual "perceived" or "central" wavelength of light, i.e. the curve showing which wavelength of light a detector is most sensitive to. Each sensor type acts as a filter to the incoming light, and can produce an output signal proportional to the amount of its central wavelength absorbed.

The wavelength signal perceived is a function of many things:

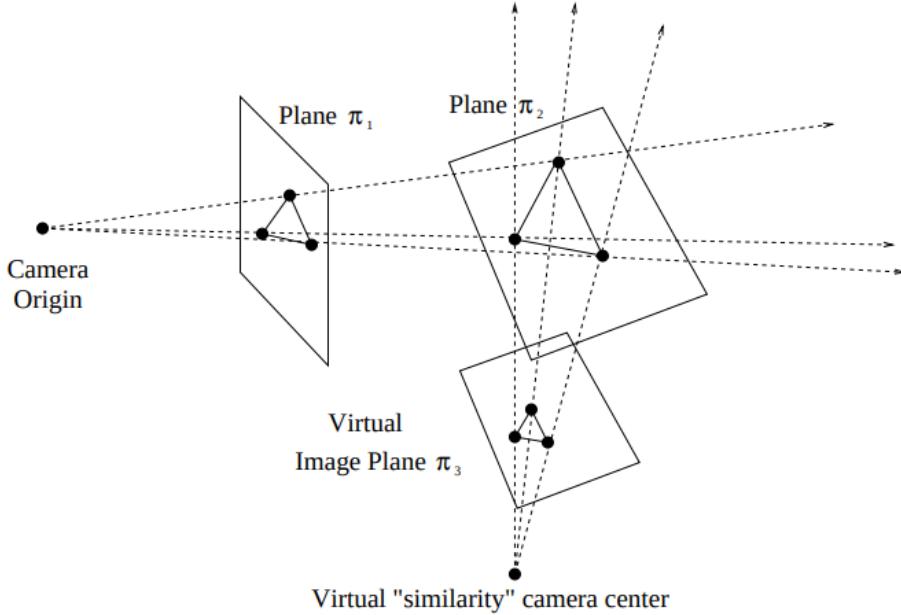
- type of source light
- the reflective properties of the objects in the scene
- the sensor detector curve

As such knowing the "real" wavelength of the light is very difficult.



Homography

An invertible linear transformation \mathbf{P} that maps points from one plane to another (think of it as a change of POV)



Given at least 4 corresponding points on each plane defining a POV, we can perform a least-square estimation of \mathbf{P} :

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \quad (6)$$

let $\vec{p} = (p_{11}, p_{12}, p_{13}, p_{21}, p_{22}, p_{23}, p_{31}, p_{32}, p_{33})$

let $\mathbf{A}_i = \begin{bmatrix} 0 & 0 & 0 & -u_i & -v_i & -1 & y_i u_i & y_i v_i & y_i \\ u_i & v_i & 1 & 0 & 0 & 0 & -x_i u_i & -x_i v_i & -x_i \end{bmatrix}$

construct $\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \dots \\ \mathbf{A}_N \end{bmatrix}$

Compute $\text{SVD}(\mathbf{A}) = \mathbf{UDV}'$

\vec{p} is last column of \mathbf{V} (eigenvector of smallest eigenvalue of \mathbf{A})

Then once we know the homography \mathbf{P} , then we can map (u, v) onto (x, y) using:

$$\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda \end{pmatrix} = \mathbf{P} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad (7)$$

(λ representing the fact that this coordinate is in homogenous space)

Focus problems

Focus set to one distance, and other nearby distances in focus (depth of focus). Further or closer not so well focused.

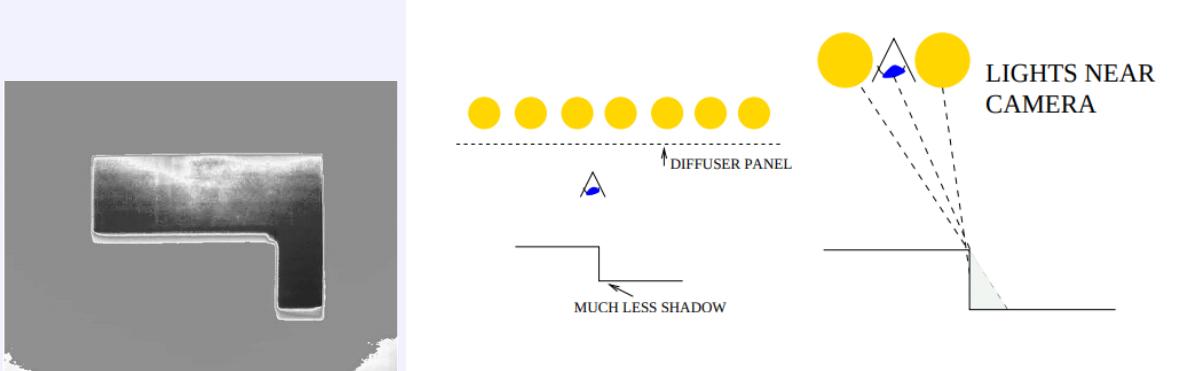


Solutions: Use smaller aperture and brighter light

Shadow problems

False colours due to different intensity of light (shadows) make it difficult to separate shapes of interest from shadows. (is the white part under this part a shadow or the edge?)

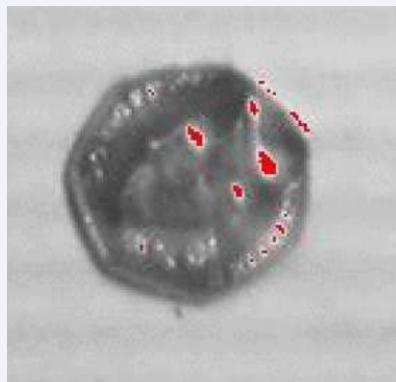
Main cause of the problem: point of light sources, the perceived brightness at a surface is proportional to the **square** of the distance between the surface and the light source.



Solutions: increase ambient lighting by using diffusing panels or lots of point lights

Specularities/highlights

(Saturated pixels set to red)



Solutions: increase ambient lighting by using diffusing panels or lots of point lights, or use smaller aperture, reduce gain and adjust gamma

Non-uniform illumination

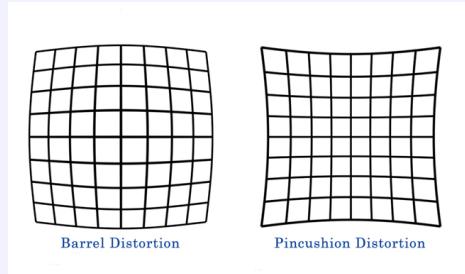
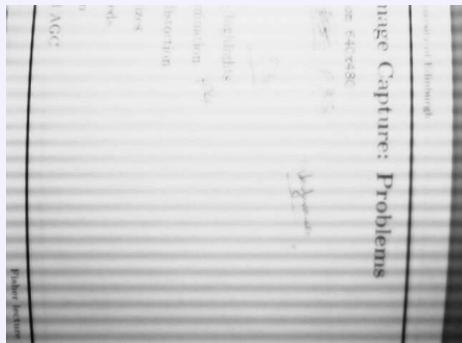
Contrast on background enhanced: may cause analysis problems



Solutions: increase ambient lighting by using diffusing panels or lots of point lights

Radial lens distortion

Lenses sometimes slightly distort the image "radially" making accurate measurements hard



Solutions: more expensive lenses, view from further away

Image Segmentation

Approaches

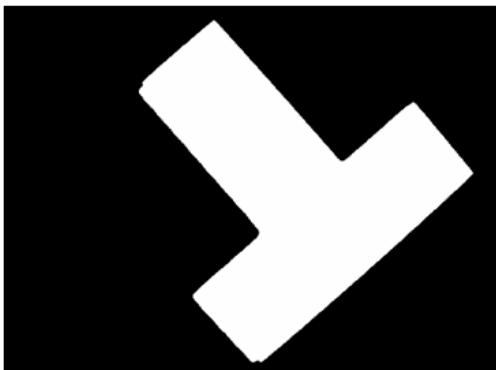
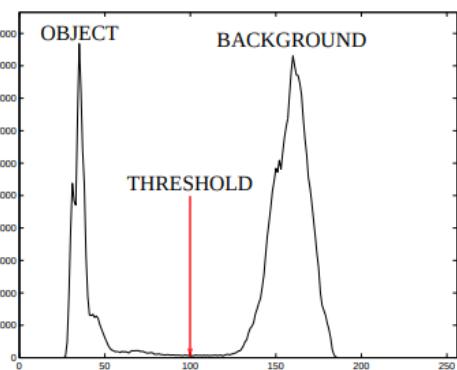
Image segmentation is the process of grouping pixels which belong together semantically, i.e. perhaps because they belong to the same object.

We can segment based on many facts:

- Contrast - objects have different lightness : use thresholding
- Change - objects different from background : background models
- Similarity - objects have consistent colours : colour clustering

Thresholding

This method assumes that pixels are separable based on their color values. We can pick threshold boundaries for each color value and select regions based on regions of pixels which fall in those boundaries.



Histogram

Thresholded Image

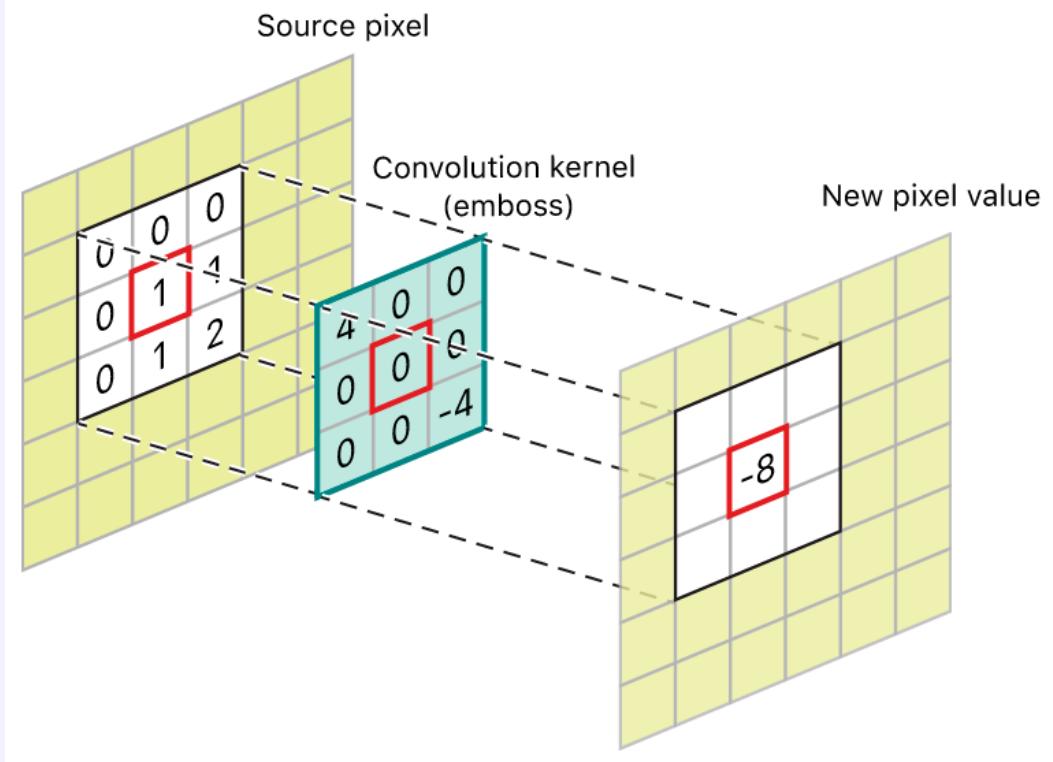
problems:

- Distributions may be broad and have some overlap leading to misclassified pixels
- variations in lighting might cause parts of the object to be missing, or shadows to be classified as objects
- color distributions might have more than 2 peaks

Convolutions

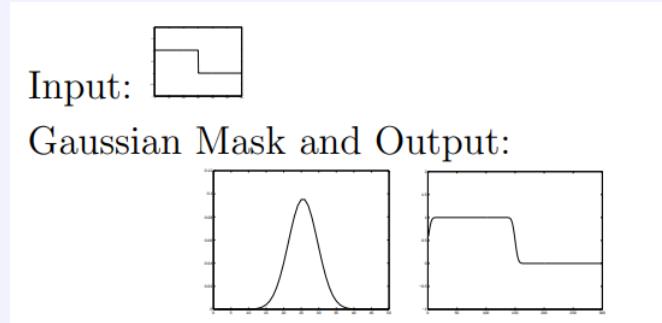
General-purpose image (and signal) processing function.
can be used to remove noise, smooth data, or detect features!

In the case of thresholding, we can use convolutions to smooth the histogram. Imagine convolutions as a sliding window, where each point in the original image is replaced with the weighted average of the window at that position with the pixels.



Convolution in 1D, with kernel of size (odd) N (even kernels require padding with zeros):

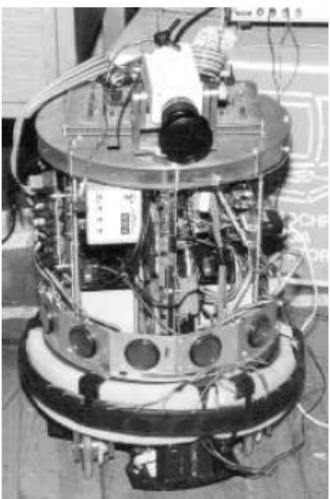
$$Output(x) = \sum_{i=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} weight(i) * input(x - i) \quad (8)$$



Convolution in 2D, with kernel of size (odd) N:

$$Output(x) = \sum_{i=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{j=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} weight(i, j) * input(x - i, y - j) \quad (9)$$

Smoothing kernel (2d gaussian)

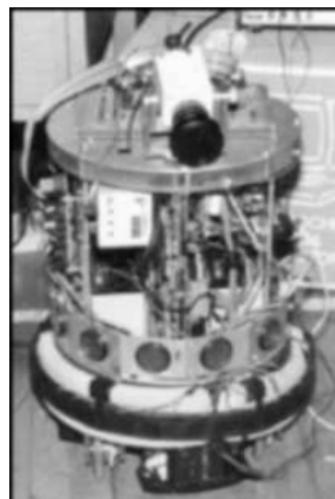


$$\frac{1}{273}$$

*

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

=



Edge Detection kernel



*

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

Edge
detection

*

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

=



=



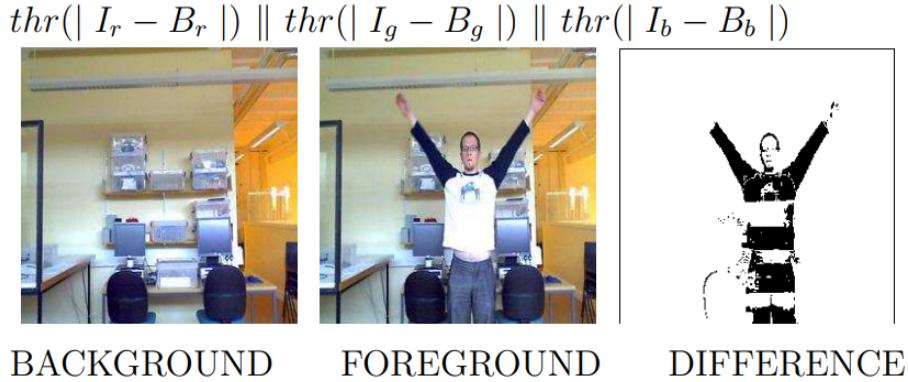
Background removal

If we have 2 images, one with just the background (**B**) and one with background and foreground (the image **I**), we can

$$N = I - B \quad (10)$$

This difference will zero-out pixels with identical values to the background, and only leave those values which are different (either positive or negative depending on if the foreground is brighter or darker than the background at each point)

We can do this for each channel of the image, and perform thresholding on the logical or between all the resulting differential pictures.



we can also use division instead of subtraction to achieve a similar effect:

$$N = I/B \quad (11)$$

This in effect removes the effects of illumination since:

$$background(i, j) = illumination(i, j) \cdot bg_reflectance(i, j) \quad (12)$$

$$object(i, j) = illumination(i, j) \cdot obj_reflectance(i, j) \quad (13)$$

The pixels with a value of 1 are going to be the background, pixels with value > 1 are lighter objects and pixels with values < 1 are darker objects (than the background)

In both of these techniques, we might need to use an operator such as the **open** operator to remove noise artifacts (with values which are just around the values which signify background pixels but not quite)

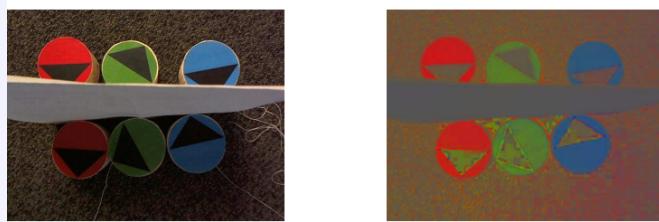
Neither will work well when the background in **I** and **B** varies wildly.

RGB Normalisation

differences in lighting can be dealt with by normalising the RGB values of the image:

$$(r', g', b') = \left(\frac{r}{r + g + b}, \frac{g}{r + g + b}, \frac{b}{r + g + b} \right) \quad (14)$$

since multiplying all values r,g,b in the original space by a constant, changes the brightness of the color, we remove this effect thanks to the equation above, mapping all different brightness values of the same colour to one value.

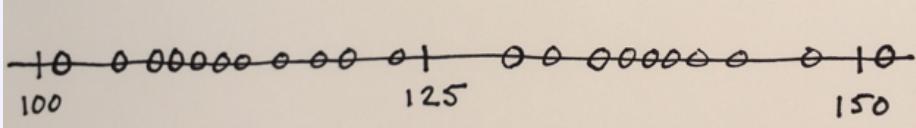


Mean Shift Segmentation

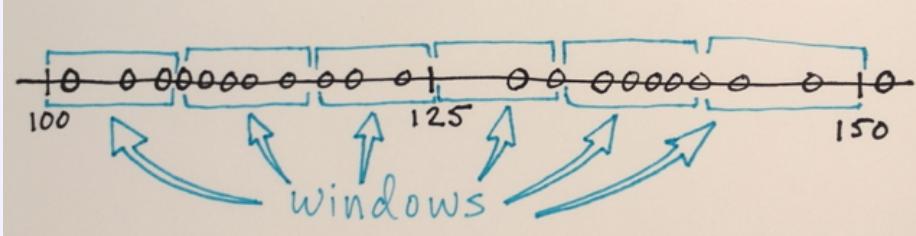
We can segment the image by performing clustering on the pixels by their color values (or any attributes for that reason)!

The algorithm works as follows:

1. create a feature space over the attributes chosen to represent each pixel (for example for a grayscale this could be a 1d intensity axis)

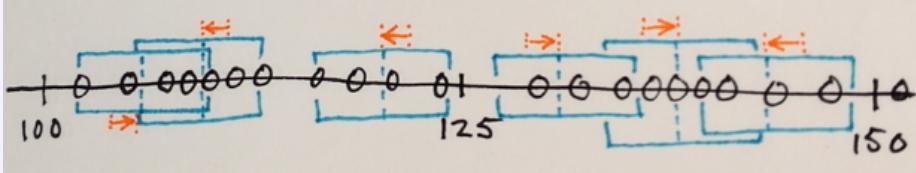


2. distribute a number of "search windows" or kernels over the space

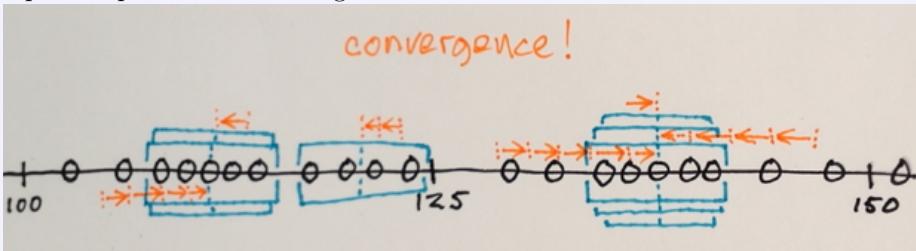


3. calculate each window's mean

4. shift the center of each window to its mean

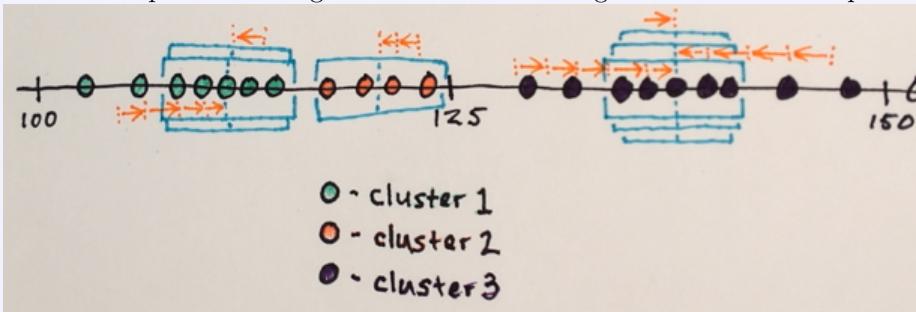


5. repeat steps 3-4 until convergence



6. merge windows ending up in close-enough locations, and call these the clusters

7. cluster each pixel according to which cluster its original window ended up at



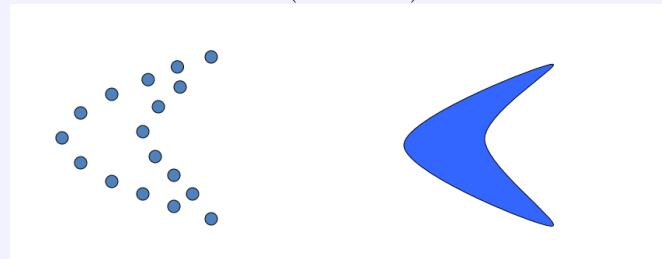
the feature space can contain any number of dimensions, and so we could include spatial, color, texture-data, and so on. This is a very versatile algorithm. It is application-independent, model-free (does not assume any shape of clusters), only requires a single parameter (window size h) which affects the scale of the clustering. It is robust to outliers and finds a variable number of modes given the same h .

The output is heavily dependent on the window size h , however. And the selection of h is not trivial. The whole algorithm is rather expensive and does not scale well with the dimension of the feature space.

Description of Segments

Shape

a set of points in the plane, or a continuous outline (silhouette)

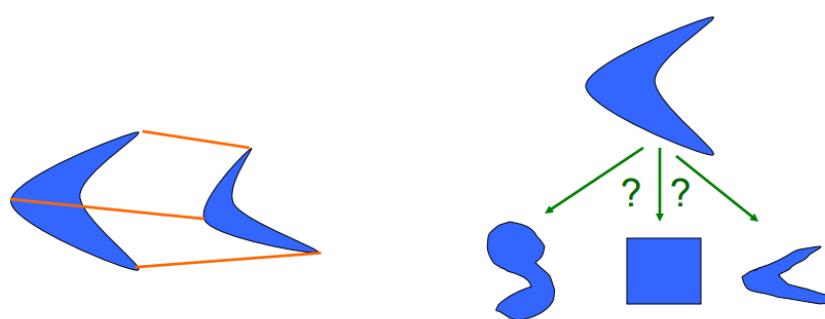


Cues

shapes can give us cues (**interior** and **boundary** cues) about the objects they outline.
Some classes are defined purely by the boundary of the shape, some are defined purely by the **contents/interior** of the shape (i.e. texture,color), and some are defined by a mixture of both

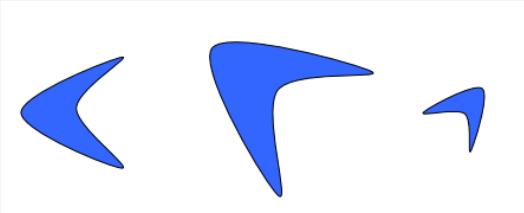
Correspondence and recognition

We can draw conclusions about similarities between shapes using **point-to-point** correspondences or **shape characteristics** to help us recognize objects belonging to certain classes.

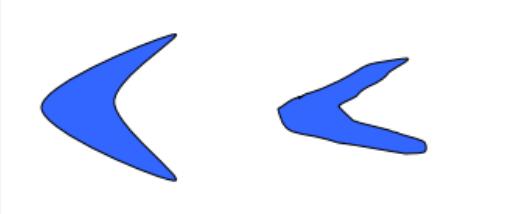


Good methods of finding similarities will be :

- Invariant to rigid transformations like: translation, rotation and scale



- Tolerant to non-rigid deformations



Global shape descriptors

Shape descriptors which put a number of a certain characteristic of a shape based on its **entirety** - hence "global".

Convexity

Convexity describes the ratio of a shape's convex hull to its perimeter, values of 1 mean that the shape is entirely convex, and values < 1 mean the shape is less convex.

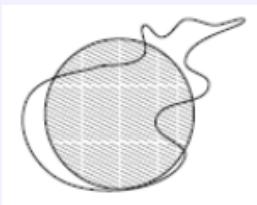


$$conv = \frac{P_{hull}}{P_{shape}} \quad (15)$$

Compactness

Compactness describes how close the perimeter of the shape is to the perimeter of the circle with the same area.

- If the circle of equal area has a smaller perimeter, this value will be smaller than 1, meaning that the shape's "mass" is distributed in a less compact manner.
- If the circle of equal area has equal perimeter, this value will be equal to 1, meaning the shape's "mass" is distributed as compactly as possible.
- This value cannot be greater than 1, as the circle is the most compact distribution of mass



$$comp = \frac{2\sqrt{A\pi}}{P_{shape}} \quad (16)$$

Elongation

The elongation is simply the ratio of the principal axes, i.e. the aspect ratio of a shape, this value can be anywhere between 0 (flat line) and ∞ (also flat line). This can be computed by taking the cross product of the principal axes with their length being set to the eigen values of the covariance matrix (if you treat each pixel as a data point)



$$elong = \frac{c_{yy} + c_{xx} - \sqrt{(c_{yy} + c_{xx})^2 - 4(c_{xx}c_{yy} - c_{xy}^2)}}{c_{yy} + c_{xx} + \sqrt{(c_{yy} + c_{xx})^2 - 4(c_{xx}c_{yy} - c_{xy}^2)}} \quad (17)$$

Properties of these global descriptors

- + Invariant to translation/rotation/scale (rigid)
- + Robust to shape deformations (non-rigid)
- + Simple
- + Fast to compute
- These do not find any point correspondences,
- Little power to discriminate between shapes (Can you discriminate between the shape of a horse and a plane with these?)

Moments

Moments in mathematics are measures which put a number on the function of interest's graph. A shape can be thought of like the graph of some function defined on the 2D space ($f(x,y)$)

Family of stable **binary** (and grey level) shape descriptions which can be made invariant to translation, rotation and scaling

Let p_{yx} be the pixel value $\in 0, 1$ at row y and column x

$$\text{Area } A = \sum_y \sum_x p_{yx}$$

Center of mass $(\hat{y}, \hat{x}) = (\frac{1}{A} \sum_y \sum_x y \cdot p_{yx}, \frac{1}{A} \sum_y \sum_x x \cdot p_{yx})$ i.e. average of x and y values weighted by "mass"

Translation invariant

let $u, v \in \mathbb{Z}$

then a family of 'central' (translation invariant) moments can be defined as:

$$m_{uv} = \sum_y \sum_x (y - \hat{y})^u (x - \hat{x})^v p_{yx} \quad (18)$$

notice how with $u, v = 2$ this is somewhat similar to variance and a little close to the moment of inertia ($\sum_p mr^2$).

This moment encapsulates the distribution of points around the center of mass, thanks to this it does not matter where the shape is positioned.

Scale invariant

We can make this family of moments invariant by noticing the fact that if we double the dimensions uniformly, then the moment m_{uv} increases by a factor of $2^u 2^v$ w.r.t weightings $(y - \hat{y}, x - \hat{x})$ and its area increases by 4. Hence $A^{\frac{u+v}{2}+1}$ grows by a factor of $4 \cdot 2^u 2^v$, Therefore the ratio:

$$\mu_{uv} = \frac{m_{uv}}{A^{\frac{u+v}{2}+1}} = \frac{m_{uv}}{m_{00}^{\frac{u+v}{2}+1}} \quad (19)$$

is invariant to scale (it cancels out the effects of increasing area, i.e. area = 1)

Rotation invariant

We can generate a similar moment using complex numbers and multiple scale-invariant moments which is invariant to rotation:

$$\text{let } c_{uv} = \sum_y \sum_x ((y - \hat{y}) + i(x - \hat{x}))^u ((y - \hat{y}) - i(x - \hat{x}))^v p_{yx}$$

then let:

$$\begin{aligned} s_{11} &= c_{11}/A^2 \\ s_{20} &= c_{20}/A^2 \\ s_{21} &= c_{21}/A^{2.5} \\ s_{12} &= c_{12}/A^{2.5} \\ s_{30} &= c_{30}/A^{2.5} \end{aligned} \quad (20)$$

we can combine these to get rotation invariant descriptors in similar magnitudes like so:

$$\begin{aligned} ci_1 &= \text{real}(s_{11}) \\ ci_2 &= \text{real}(10^3 \cdot s_{21} \cdot s_{12}) \\ ci_3 &= 10^4 \cdot \text{real}(s_{20} \cdot s_{12}^2) \\ ci_4 &= 10^4 \cdot \text{imag}(s_{20} \cdot s_{12}^2) \\ ci_5 &= 10^6 \cdot \text{real}(s_{30} \cdot s_{12}^3) \\ ci_6 &= 10^6 \cdot \text{imag}(s_{30} \cdot s_{12}^3) \end{aligned} \quad (21)$$

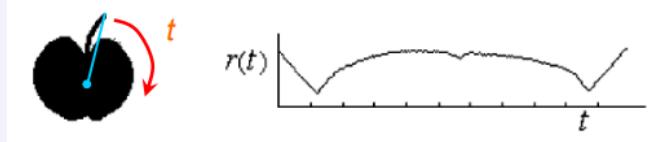
Shape signatures

We can represent the shape using a 1D function ($f(t)$) defined via the points on the boundary of the shape. Once we have such descriptors, we can establish similarity between two shapes using: $\int f(t) - f(t') \, dt$ i.e. the difference between the shape's descriptors integrated over t

Centroid distance

for angle t , and point on boundary at that angle $p(t)$

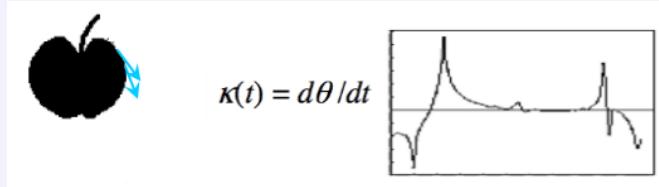
$$r(t) = d(p(t), \text{centroid}) \quad (22)$$



Curvature

for angle t , and angle θ representing the angle between points $p(t)$ and $p(t + \Delta t)$ on the boundary at the angles t and $t + \Delta t$

$$k(t) = d\theta/dt \quad (23)$$



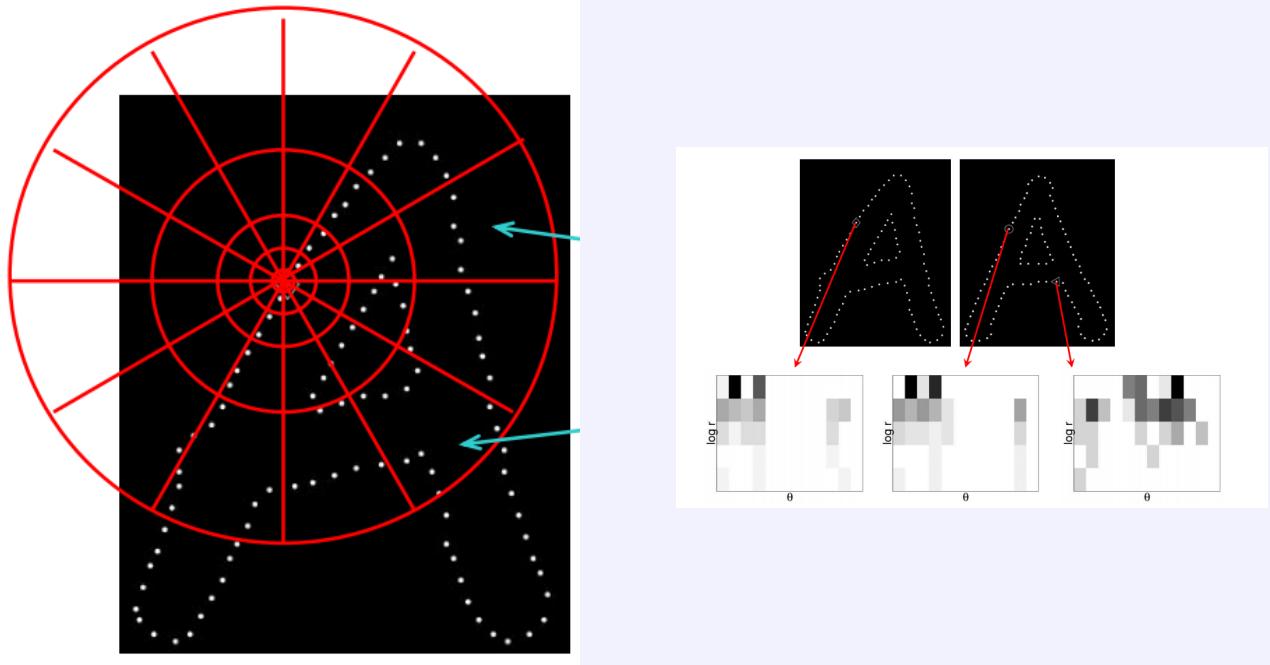
Properties of shape signatures

- + invariant to translation, scale (if shape is normalized), rotation (if orientation is normalized)
- + point correspondences (if both descriptors are aligned)
- + informative
- + deformations affect signature locally and not globally (i.e. at a single point of the signature)
- ~ manages to handle shape deformation to some degree
- where to start t ? high computational cost of alignment of two signature functions
- sensitive to noise (especially with derivatives)

Shape Context

Shape context is a shape descriptor utilizing the local properties of points on the boundary of each shape to establish **point-to-point** correspondences

We do this by counting the number of other points around the points on the boundary of each shape in each bin of a polar-coordinate "kernel" (this forms a histogram)



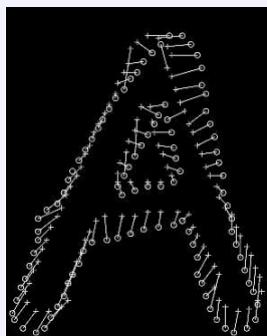
We can compare the K-bin histograms $h_i(k), h_j(k)$ of two points i, j on different shapes respectively, using the chi-squared test:

$$C(i, j) = \frac{1}{2} \sum_{k=1}^K \frac{(h_i(k) - h_j(k))^2}{h_i(k) + h_j(k)} \quad (24)$$

This establishes a cost function over which we can pair-up the corresponding points on each shape, by finding the least-cost matching $\pi(p)$ of points on one shape to the other (perhaps using the hungarian or blossom algorithms) which minimizes the total cost:

$$H(\pi) = \sum_{p \in \text{all_points}} C(p, \pi(p)) \quad (25)$$

thus establishing a point-to-point correspondence between two shapes:



Properties of shape signatures

- + invariant to translation
- + invariant to scaling (if we normalize the radial distances between points in each shape by their mean)
- + informative - describes points in the context of the overall shape
- + handles non-rigid deformations quite well - more sensitive for deformations closest to the point of interest due to shape of kernel
- not invariant to scale (but could be added by measuring angles in terms of tangents at each point instead of global coordinates)
- many parameters (# and size of bins, # of iterations, # number of points, etc..)
- very expensive computationally

Object recognition

Assumptions & Approaches

Approaches

Several approaches to classification/recognition. Choose the same class as objects with:

- **Shape** - similar shape descriptors
- **Appearance** - similar pixel values
- **Geometric** - similar structures in similar places with similar parameters
- **Graph** - similar part relationships
- **Bag of words** - similar local feature descriptors (frankenstein objects made up of smaller objects)

Assumptions

Assumptions made in this course:

- Flat objects, viewed orthographically
- Always looked at from same distance
- Good contrast everywhere
- No specularities
- shape-based recognition only

Shape-based recognition

1. Extract object from image via segmentation
2. Compute its properties
3. Use those properties to compute the class it belongs to
4. Learn/improve the model properties for the classes

Probabilistic object recognition

The process of classifying the shape into a class by calculating the probability of it belonging to each class.

Bayes rule

we can calculate the probability of feature vector \vec{x} (which may be a collection of shape descriptor values, or any other properties) being drawn from the probability distribution which best describes the class c as:

$$p(c|\vec{x}) = \frac{p(\vec{x}|c)p(c)}{p(\vec{x})} = \frac{p(\vec{x}|c)p(c)}{\sum_k p(\vec{x}|k)p(k)} \quad (26)$$

where:

- $p(\vec{x}|c)$ is the probability of observing the feature vector \vec{x} if it belongs to class c (using the distribution of feature vectors from class c)
- $p(c)$ is the *a priori* probability of observing a feature vector from class c (before making any observations)
- $p(\vec{x})$ is the total probability of seeing the feature vector \vec{x} amongst all the classes

Multivariate Gaussian distribution

how do we model the probability $p(\vec{x}|c)$ of observing each feature class, knowing some feature vectors belonging to each class ?

We can perform Maximum likelihood estimation (MLE) on the observed $k > n$ (n being the dimensionality of \vec{x}) "training" instances of data and build a multivariate gaussian distribution for each class. MLE yields the following values for each class:

- mean vector of each feature \vec{m}_c of dimension n - average value of each feature in class c:

$$\vec{m}_c = \frac{1}{k} \sum_{i=1}^k \vec{x}_i \quad (27)$$

- covariance matrix \mathcal{A}_c - the $n \times n$ matrix of co-variances between each pair of features/properties:

$$\mathcal{A}_c = \frac{1}{k-1} \sum_{i=1}^k (\vec{x}_i - \vec{m}_c)(\vec{x}_i - \vec{m}_c)^T \quad (28)$$

With those properties the multivariate gaussian is formed as follows:

$$p(\vec{x}|c) = \frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{|\mathcal{A}_c|^{\frac{1}{2}}} \exp^{-\frac{1}{2}[(\vec{x}-\vec{m}_c)^T \mathcal{A}_c^{-1} (\vec{x}-\vec{m}_c)]} \quad (29)$$

Recognition Algorithmics

We split the data into:

- a **training** set to estimate the model's parameters (e.g. the gaussian distributions)
- a **validation** set to pick the ideal "hyper" parameters which affect the performance of the algorithm without necessarily affecting the underlying model
- a **test** set to evaluate the performance of the algorithm

note: *we must have more training samples than the dimensions of the feature vectors used!*

Chamfer-based shape matching

We can employ an entirely different method of object recognition. In a way similar to 2D convolutions:

- Extract edges/contours of image we want to identify object in (perhaps using convolutions)
- Create a chamfer or "template" which forms the shape of the object we want to identify in the image
- Slide it over the image, at each point find the "chamfer distance" by calculating the average distance of points on the chamfer to closest edges in the image:

$$D_{chamfer}(T, I) = \frac{1}{|T|} \sum_{t \in T} d_I(t) \quad (30)$$

where:

T, I are the sets of template and image points respectively

$d_I(t)$ is the minimum distance for any template point t to any point in the image I

$|T|$ is the number of points in the template

Optimisations

the naive implementation is very expensive as we re-compute the distances between each time. Instead we can do this only once by producing a look-up image of distances encoding the distance between each pixel in the image to the nearest edge inside it:

