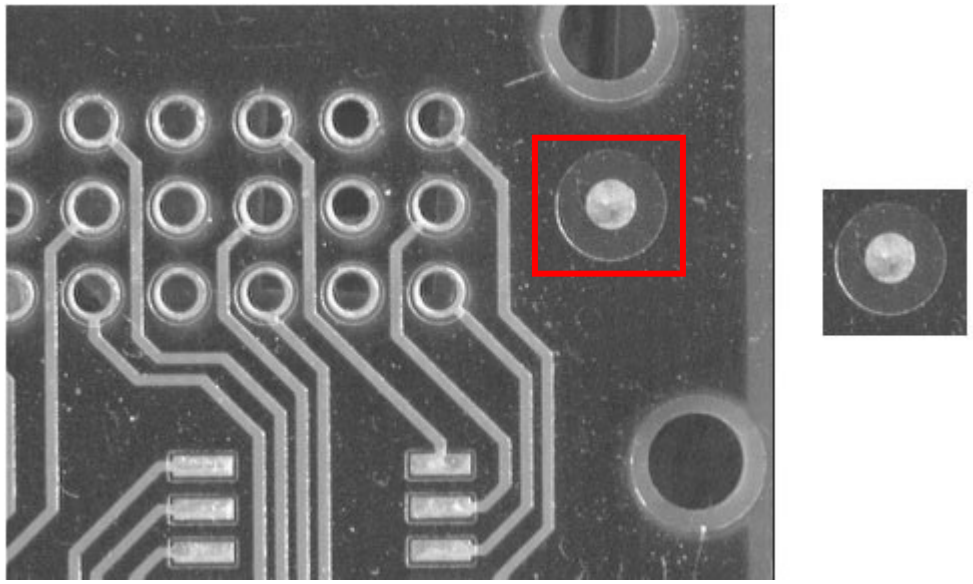
Image Processing and Computer Vision

Prof. Giuseppe Lisanti
giuseppe.lisanti@unibo.it

Instance-level Object Detection

it is different from class-level object detection

- The *Instance-level Object Detection* problem occurs in countless applications and can be formulated as follows.
  - Given a reference image (aka model image) of a *specific object*, determine whether the object is present or not in the image under analysis (aka target image) and, in case of detection, estimate the pose of the object.



- Depending on the application, the pose may often be given by a translation, a roto-translation or a similarity (roto-translation plus scale)

Instance-level Object Detection

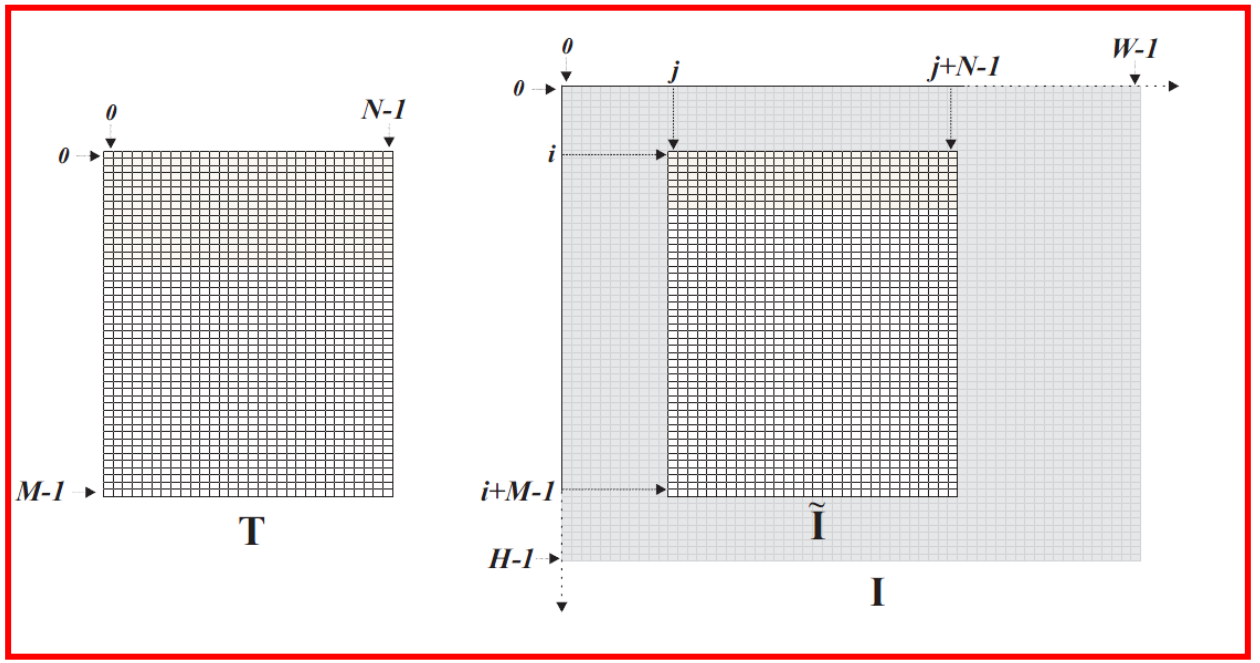
- The problem has a number of diverse facets: the sought object may appear only once or multiple times in the target image, we may be interested in detecting a number of diverse objects, each of which, in turn, may appear once of multiple times.
  - For the sake of simplicity, and without much loss of generality, we will refer mainly to the basic setting: *detecting a single object that may appear once in the target image*. Generalization to the other variants turns out more often than not straightforward.
- Typical nuisances to be dealt with are *intensity changes, occlusions and clutter*.
- Computational efficiency is a major requirement in most practical applications.
- This problem is characterized by a *limited variability* as the assumption deals with the appearance of the object being captured by a single model image (i.e. roughly planar objects or no view-point changes) and the pose is typically either a 2D translation or a 2D roto-translation or a similarity.
- Given the limited variability, the problem can been addressed successfully by classical computer vision techniques, the major applications dealing mainly with the space of industrial vision.

- Instead, *Category-level Object Detection* aims at detecting certain kind of object(s) (e.g. cars, pedestrians..) regardless of their appearance and pose. Due to the high-variability, this problem is addressed by machine/deep learning techniques.

Template Matching

- The model image is slid across the target image to be compared at each position to an equally sized window by means of a suitable (dis)similarity function

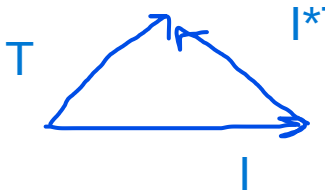
(Dis)Similarity Functions



is the sub-image with the same dimension m,n of the template

- Both $\tilde{I}(i,j)$, the window at position (i, j) of the target image having the same size as T, as well as T can be thought of as M·N-dimensional vectors (flatten)
- Compute pixel-wise intensities differences: $SSD(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(I(i+m, j+n) - T(m,n)\right)^2$

(Dis)Similarity Functions



I and T are vectors, where each pixel intensity value corresponds to a dimension in the high-dimensional space

- Sum of Absolute Differences: $$SAD(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left|I(i+m, j+n) - T(m,n)\right|$$

  faster than SSD

because they both compute difference of intensities, and if intensity differs too much from T and I, it will result in an inaccurate matching

with controlled light conditions

- Are SSD and SAD invariant to intensity changes? NO, why? when we should use them?

- Normalised Cross-Correlation: $$NCC(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n)\cdot T(m,n)}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n)^2}\cdot\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T(m,n)^2}}$$

  Similarity!

  it is a tecnique to mitigate the intensity-change issue, resulting in a more robust approach

  Normalization helps reduce the influence of overall brightness differences between the template and the patch.

  - Represents the cosine of the angle between vectors I(i,j) e T $$NCC(i,j) = \frac{\tilde{\mathbf{I}}(i,j)\cdot\mathbf{T}}{\left\|\tilde{\mathbf{I}}(i,j)\right\|\cdot\|\mathbf{T}\|} = \frac{\left\|\tilde{\mathbf{I}}(i,j)\right\|\cdot\|\mathbf{T}\|\cdot\cos\theta}{\left\|\tilde{\mathbf{I}}(i,j)\right\|\cdot\|\mathbf{T}\|} = \cos\theta$$

    dot product can be represented in this way

    cosine similarity between two vectors measures how closely their directions align in this high-dimensional space

  - Invariant to linear intensity changes $$\tilde{\mathbf{I}}(i,j) = \alpha\cdot\mathbf{T}$$

    NCC it's invariant to a linear intesity change but if we consider the whole affine intensity function and we add a bias, the vector T changes direction

(Dis)Similarity Functions

ZNCC aims to find how well a template (T) matches a specific patch (I) within a larger image by considering not just the difference in pixel intensities, but also the overall intensity variations.

- Zero-Mean Normalised Cross-Correlation, Correlation Coefficient

$$\mu(\tilde{I}) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n) \qquad \mu(T) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T(m,n)$$

1) Calculate the mean intensity (average) of all pixels in the template (T_mean) and the image patch (I_mean).

2) centering the data around zero, subtracting the respective means, ensuring the data is distributed around zero, reducing the influence of overall brightness differences between the template and the patch.

subtracting the mean from each pixel value within the template and image patch

- The NCC is computed after subtraction of the means:

Mean of the subimage

$$I(i+m, j+n) \rightarrow \left(I(i+m, j+n) - \mu(\tilde{I})\right)$$

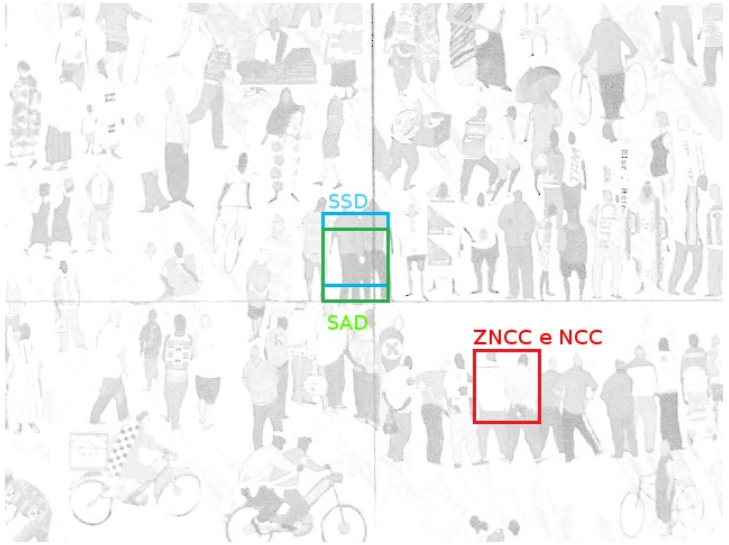
$$T(m,n) \rightarrow \left(T(m,n) - \mu(T)\right)$$

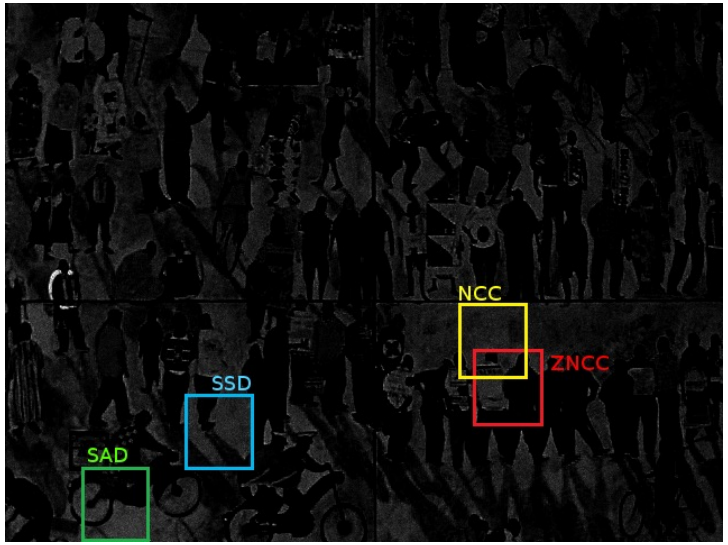
$$ZNCC(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(I(i+m, j+n) - \mu(\tilde{I})\right)\cdot\left(T(m,n) - \mu(T)\right)}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(I(i+m, j+n) - \mu(\tilde{I})\right)^2}\cdot\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(T(m,n) - \mu(T)\right)^2}}$$

- Invariant to affine intensity changes $\tilde{\mathbf{I}}(i,j) = \alpha\cdot\mathbf{T} + \beta$

The mean intensity reflects the overall brightness or darkness of the template/patch. Subtracting the mean removes this bias, allowing the NCC calculation to focus on the variations in intensity that contribute to the object's shape or pattern.

Comparison under significant intensity changes





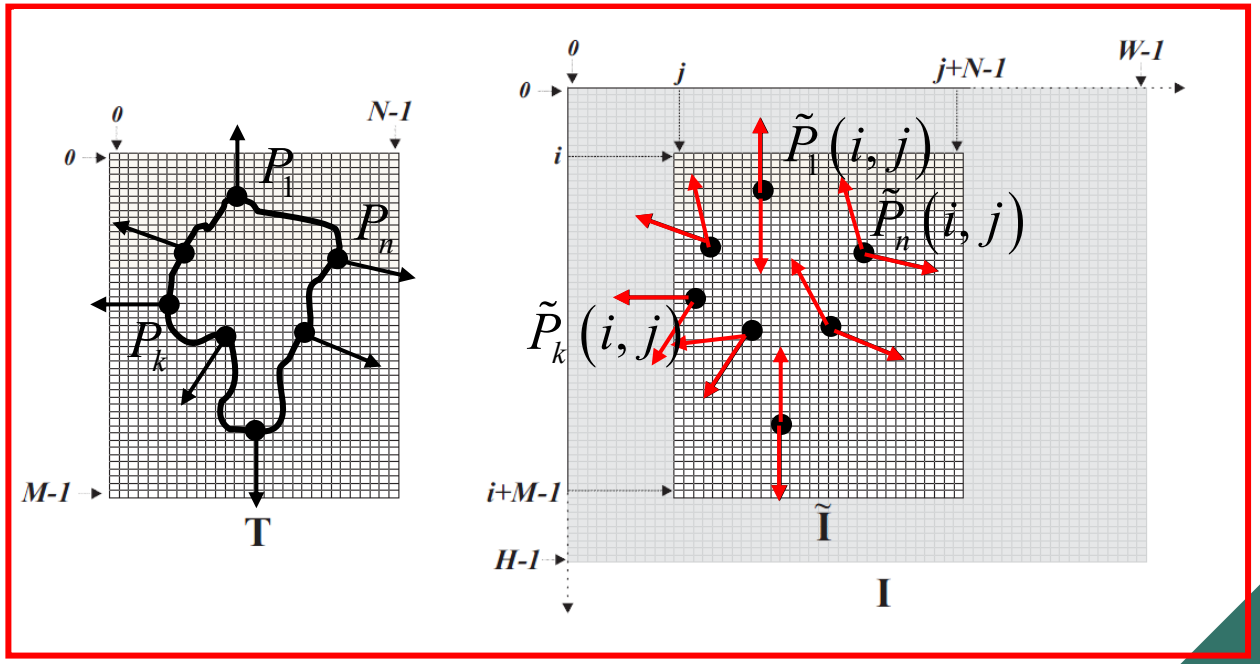
ZNCC turns out to be a similarity function very robust to intensity changes!

Shape-based Matching

- Edge-based template matching approach
  - First, a set of control points, P_k, is extracted from the model image by an Edge Detector and the gradient direction at each P_k is stored
    - Template composed by offsets and gradient directions
  - Then, at each position (i,j) of the target image, the **recorded** gradient directions associated with control points are compared to those at their corresponding image points, $P_k(i,j)$

- The more the red arrows are aligned to the black arrows the more similar the sub-image is to the template

- We do not perform edge detection on the target image, just on the template



Steger, 2002

Similarity Function

between gradients' direction

we do care only about the direction, so we NORMALIZE the gradient to a unit vector

Normalized => unit vector

$$\mathbf{G}_k(P_k) = \begin{bmatrix} I_x(P_k) \\ I_y(P_k) \end{bmatrix}, \mathbf{u}_k(P_k) = \frac{1}{\|\mathbf{G}_k(P_k)\|} \begin{bmatrix} I_x(P_k) \\ I_y(P_k) \end{bmatrix}, k = 1..n$$

Template

gradient magnitude

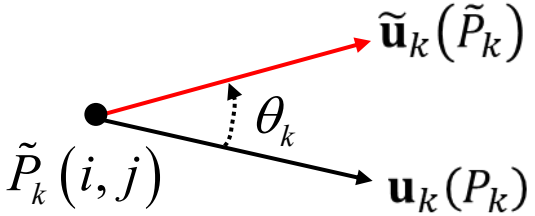
$$\tilde{\mathbf{G}}_k(\tilde{P}_k) = \begin{bmatrix} I_x(\tilde{P}_k) \\ I_y(\tilde{P}_k) \end{bmatrix}, \tilde{\mathbf{u}}_k(\tilde{P}_k) = \frac{1}{\|\tilde{\mathbf{G}}_k(\tilde{P}_k)\|} \begin{bmatrix} I_x(\tilde{P}_k) \\ I_y(\tilde{P}_k) \end{bmatrix}, k = 1..n$$

Target

compute dot product

$$S(i,j) = \frac{1}{n}\sum_{k=1}^{n} \mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k) = \frac{1}{n}\sum_{k=1}^{n} \cos\theta_k$$

- The similarity function spans the interval [-1; 1]
  - It takes its maximum value when all the gradients at the control points in the current window of the target image are perfectly aligned to those at the control points of the model image
  - If they are perfeclty aligned the angle is zero, the cosine is 1, the sum is n, which divided by n is 1, and vice versa



- Choosing a detection threshold, S_{min}, can be thought of as specifying the fraction of model points which must be seen in the image to trigger a detection

it basically defines how many points we need to trigger a matching between the template and the target image

More robust similarity functions

we want to be robust againts polarity inversions

- Certain application settings call for invariance to *global inversion of contrast polarity* along object's contours, as the object may appear either darker or brighter than the background in the target image

- This kind of invariance can be achieved by a slight modification to the similarity function defined previously (global)

$$S(i,j) = \frac{1}{n}\left|\sum_{k=1}^{n} \mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k)\right| = \frac{1}{n}\left|\sum_{k=1}^{n} \cos\theta_k\right|$$

compute absolute value disregarding the sign, the polarity

- The following function is even more robust due to the ability to withstand local contrast polarity inversions (local)

$$S(i,j) = \frac{1}{n}\sum_{k=1}^{n}\left|\mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k)\right| \quad = \frac{1}{n}\sum_{k=1}^{n}|\cos\theta_k|$$

a way to find SPECIFIC SHAPES

The Hough Transform

it takes as input the output of an edge detector

a way to express shapes with an analytical equation
- it works in the space of the parameters defined by the analytical equation

- The *Hough Transform (HT)* enables to detect objects having a known shape that *can be expressed by an equation* (e.g. lines, circles, ellipses..) based on projection of the input data into a suitable space referred to as *parameter* or *Hough space* (different from the image space)

In this space, each point represents a possible way to draw the target shape

- The HT turns a global detection problem into a local one (i.e., look for feature points into the parameter space, instead of looking for the whole shape in the image space)
- The HT is usually applied after an edge detection process (i.e., the actual input data consist of the edge pixels extracted from the original image)
- The HT is robust to noise and allows for detecting the sought shape even though it is partially occluded into the image (up to a certain user-selectable degree of occlusion)

- The HT was invented to detect lines and later extended to other analytical shapes (circle, ellipses) as well as to *arbitrary shapes => Generalized Hough Transform (GHT)*
- The GHT principle is widely deployed also within object detection pipelines relying on local invariant features such as, e.g., SIFT

Basic Principle

we are transforming the problem of detecting shapes to the problem of detecting point instersections in the parameter space (which is way easier)

- HT formulation for lines, what is fixed and what is changing in this equation? $y - mx - c = 0$

- In the usual image space interpretation of the line equation the parameters $(\hat{m}, \hat{c})$ are fixed

slope and intercept

the parameters m and c of the line equation are fixed, and you're finding the image points that belong to lines with those specific parameters

$$y - \hat{m}x - \hat{c} = 0$$

so that the equation represents the mapping from point $(\hat{m}, \hat{c})$ of the *parameter space to the image points belonging to the line*

for each line in the image, m and c are constant

the image point

- However, we may instead fix $(\hat{x}, \hat{y})$

$$\hat{y} - m\hat{x} - c = 0$$

so the equation represents the mapping from image point $(\hat{x}, \hat{y})$ to the **parameter space** providing **all the lines through the image point** (x^, y^)

because we allow the parameters m and c, the slope and the intercept to vary

Basic Principle

- Consider two image points P_1,P_2 and map both into the parameter space, we get two lines intersecting at the parameter space point representing the image line through P_1,P_2 :

we can compute the parameters m and c of the line passing through two points in this way

$$\begin{cases} \hat{y}_1 - m\hat{x}_1 - c = 0 \\ \hat{y}_2 - m\hat{x}_2 - c = 0 \end{cases} \Longrightarrow \begin{cases} m = \frac{\hat{y}_2 - \hat{y}_1}{\hat{x}_2 - \hat{x}_1} \\ c = \frac{\hat{x}_2\hat{y}_1 - \hat{x}_1\hat{y}_2}{\hat{x}_2 - \hat{x}_1} \end{cases}$$

if P1 = (1,1) and P2 = (2,3)
general equation: y = mx + c
if we only know one: (y-y1) = m(x-x1)

solving for the parameters:
m = 3-1 / 2-1 = 2
c = 2*1 - 1*3 / 2-1 = -1

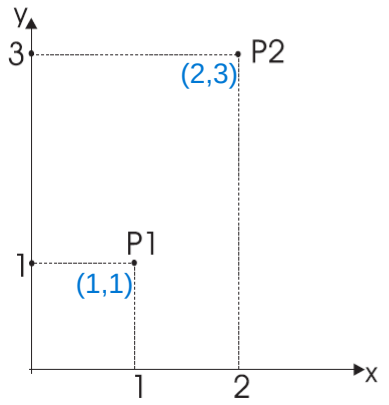
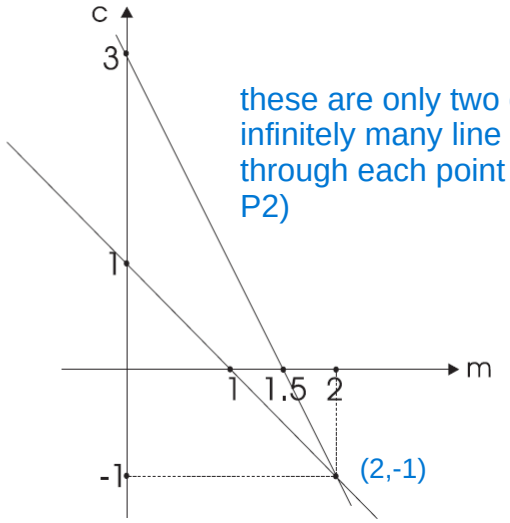


Image Space



these are only two of the infinitely many line passing through each point (P1 and P2)

Parameter Space

this intersection point represents the values of the parameters m and c of the line passing through P1 and P2

- *m* and *c,* represent the parameters of the line passing through P1 and P2

since each pair of points contributes to an intersection in the parameter space, and we have (n 2) pairs of points ...

- More generally, if we map ***n*** image points we get as many intersections as ***n(n-1)/2*** (i.e. the number of lines through the n image points)

... where (n 2) is the number of combinations of n choose 2 elements, there are (n 2) intersections, and because (n 2) = n(n-1)/2 we got that result

Basic Principle

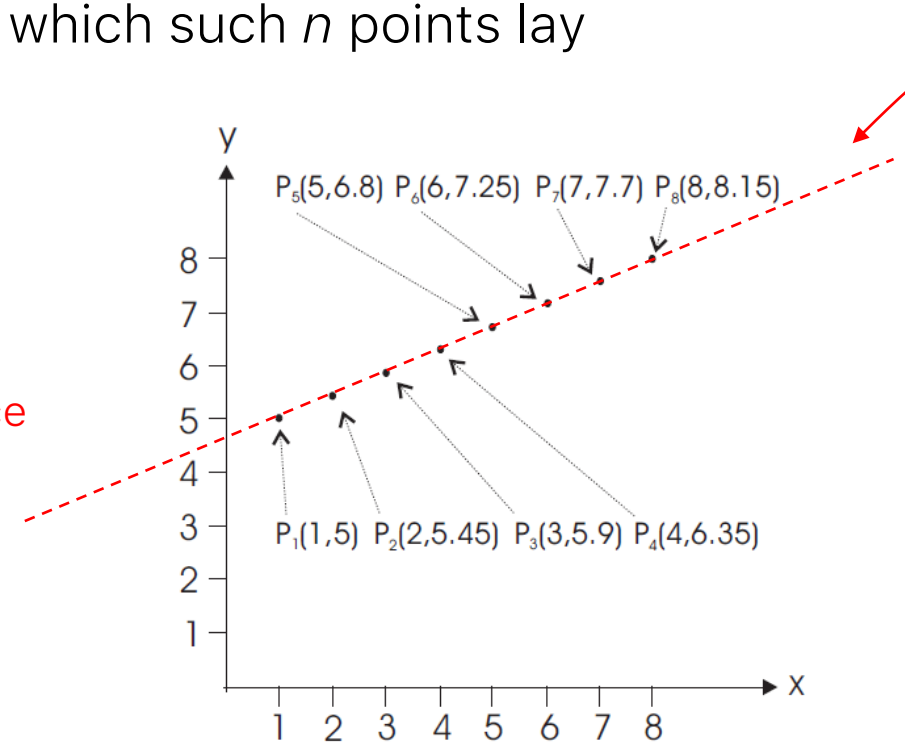
we search in the parameter space the points with the highest number of intersections

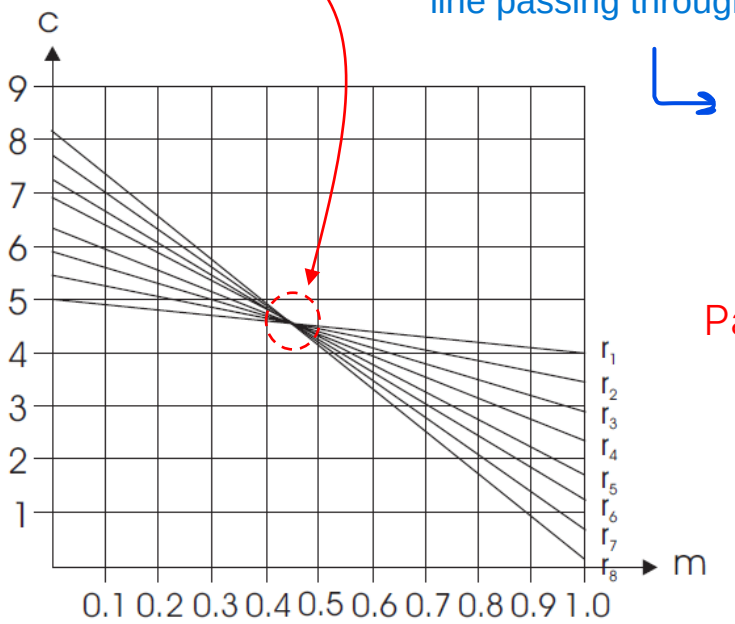
- Considering *n* collinear image points, we can notice that their corresponding transforms (i.e. parameter space lines) will intersect at a single parameter space point representing the image line along which such *n* points lay

there exists only one configuration of values for the parameters m and c such that they define a line passing through all those points

↳ if it so, then we have detected a line in our image

Image Space





Parameter Space

we first compute the intersection point, then we draw the lines passing through it

- Rather than looking at an extended shape into the image, we look for a specific feature (where lines intersect) in the parameter space of lines

Basic Principle

The transformation from image space to parameter space involves a process of voting, where each point in the image space contributes to potential shapes (lines, circles, etc.) in the parameter space. This transformation allows you to detect geometric shapes in an image, even in the presence of noise or missing parts of the shape.

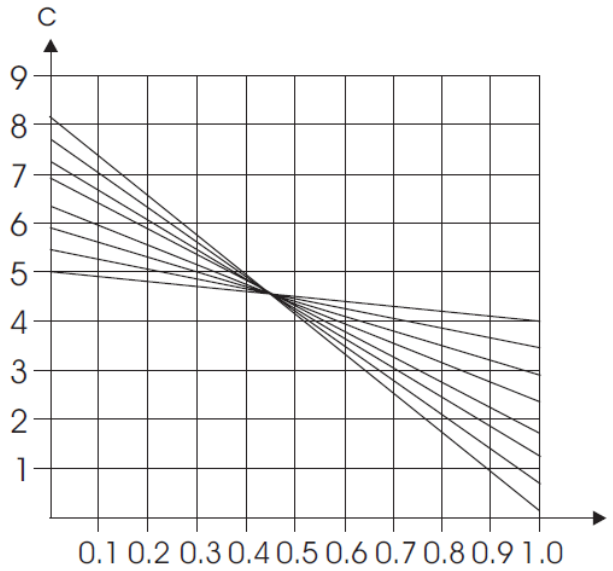
- Therefore, given a sought *analytic shape represented by a set of parameters*, the HT consists in mapping image points (i.e. usually edge points) so as to create *curves into the parameter space of the shape*
- *Intersections of parameter space curves* indicate the presence of image points explained *by a certain instance of the shape*
  - the more the intersecting *curves* the more are such image points and thus the higher is the evidence of the presence of that instance in the image
- Detecting objects through the HT consists in finding *parameter space points through which many curves do intersect* (a local rather than global detection problem)
- To make it work in practice, the parameter space needs to be quantized and allocated as a memory array, which is often refereed to as *Accumulator Array (AA)*
- Curves are “drawn” into the AA by a so called *voting process*:
  1. the transform equation is repeatedly computed to increment the bins satisfying the equation
  2. *a high number of intersecting curves at a point of the parameter space will provide a high number of votes* into a bin of the AA
  3. Finding parameter space points through which many curves do intersect is thus implemented in practice by finding *peaks of the AA*, i.e. local maxima showing a high number of votes

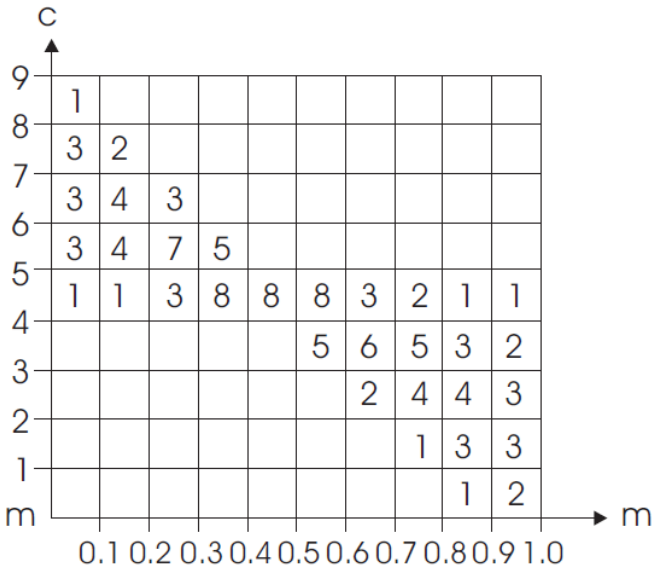
AA for line detection (Voting)

this cannot work due to occlusion

- The AA highlights the presence of a line with $m \in [0.3, 0.6],\ c \in [4, 5]$

Continuous Parameter Space



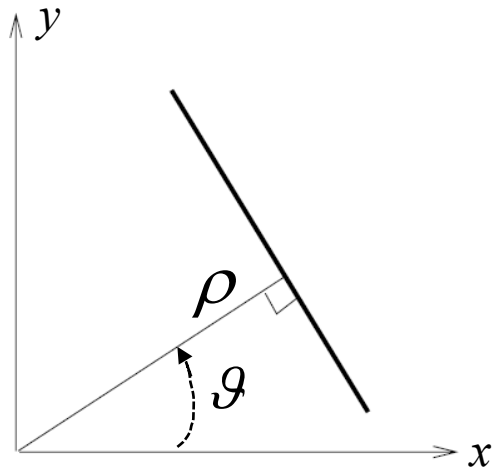


Quantized AA

- To detect the line more accurately, the AA should be quantized more finely
- The HT is robust to noise because spurious votes due to noise unlikely accumulate into a bin so as to trigger a false detection
- A partially occluded object can be detected provided that the threshold on the minimum number of votes required to declare a detection is lowered according to the degree of occlusion to be handled

HT for Line Detection

- The usual line parametrization considered so far (i.e. *y-mx-c=0*) is impractical due to *m* spanning an infinite range
- The "normal parametrization" is adopted in the HT for lines: $\rho = x\cos\vartheta + y\sin\vartheta$



- Image points $(\hat{x},\hat{y})$ are mapped into sinusoidal curves

  of the (ϑ,ρ) parameter space: $\rho = \hat{x}\cos\vartheta + \hat{y}\sin\vartheta$

- With the normal parametrization: $\vartheta \in \left[-\frac{\pi}{2},\frac{\pi}{2}\right], \quad \rho \in [-\rho_{\max},\rho_{\max}]$

- while ρ_{max} is usually taken as large as the image diagonal: $N \times N \text{ pixels} \rightarrow \rho_{\max} = N \cdot \sqrt{2}$

Generalized Hough Transform

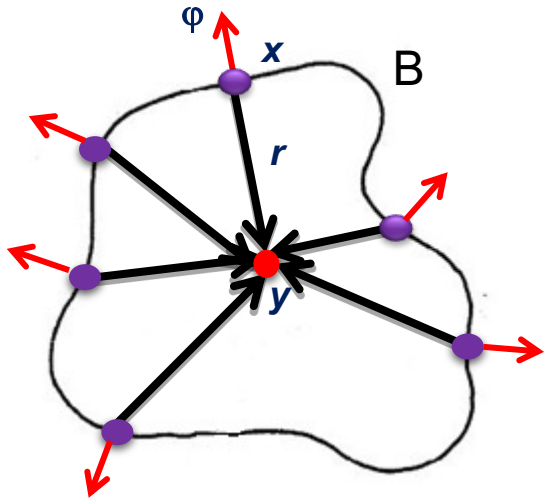
the Generalized Hough Transform provides a flexible framework for detecting arbitrary shapes or objects in images by learning from a reference model and then matching this model to features extracted from the input image

not just predefined geometric primitives like lines or circles

- The HT has been extended to detect arbitrary (i.e. non analytical) shapes:
- Off-line Phase (build the object's model)

we learn the features of the shape we want to recognize

1. A reference point y is chosen (e.g. barycentre)
2. For each point x belonging to object's border B:
    1. Compute gradient direction ϕ(x)
        1. Gradient direction is quantized according to a chosen step Δϕ
    2. Compute vector r from y to x (i.e. r = y-x).
3. Store r as a function of Δϕ (R-Table)
4. An entry in the R-Table can contain several r vectors

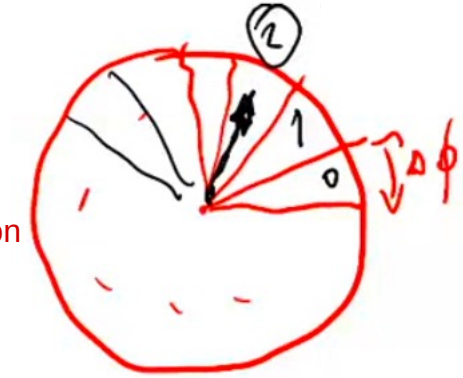


The reference model or table, often referred to as an R-table, contains information about the shape's features, their spatial relationships, and any other relevant attributes.

i	ϕ_i	R_{ϕ_i}
0	0	$\{r \mid y - r = x, x \in B, \phi(x) = 0\}$
1	$\Delta\phi$	$\{r \mid y - r = x, x \in B, \phi(x) = \Delta\phi\}$
2	$2\Delta\phi$	$\{r \mid y - r = x, x \in B, \phi(x) = 2\Delta\phi\}$
...	...	...

for each gradient direction, store several vectors r

we collect join vectors depending on the gradient direction



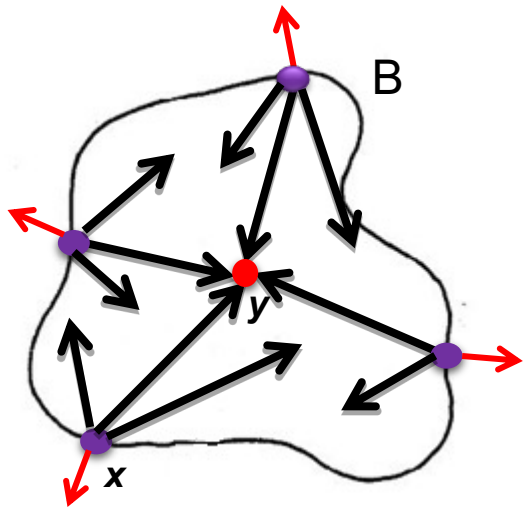
we quantize the gradient direction

Generalized Hough Transform

in this way we can find not any shape but only instances of the model-shape under interest

- On-line Phase
  1. We do edge detection first — the input image is analyzed to extract features or descriptors that characterize potential instances of the reference shape
  2. An image A[y] is initialized as accumulator array. For each edge pixel x of the input image:
     1. Compute gradient direction ϕ — The extracted features from the input image are compared against the reference table or model generated during the off-line phase
     2. Quantize ϕ to index the R-Table. For each r_i vector stored into the accessed row:
        1. Compute the position of the reference point $y=x + r_i$
        2. Cast a vote into the accumulator array A[y]++
  3. Instances of the sought object are detected by finding peaks of the accumulator array — peaks identify potential matches

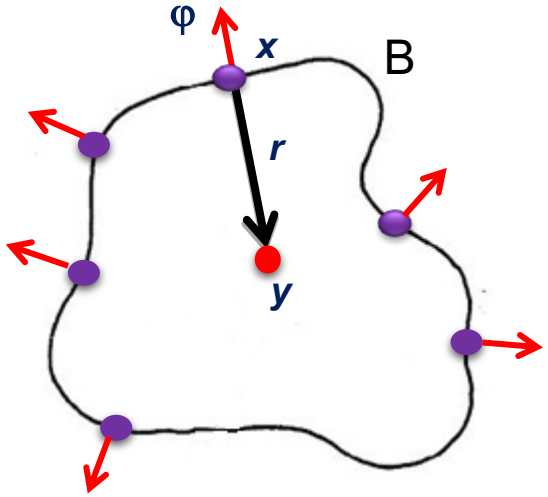
i	ϕ_i	R_{ϕ_i}
0	0	$\{r \mid y - r = x, x \in B, \phi(x) = 0\}$
1	$\Delta\phi$	**r_1, r_2, r_3**
2	$2\Delta\phi$	$\{r \mid y - r = x, x \in B, \phi(x) = 2\Delta\phi\}$
...	...	...

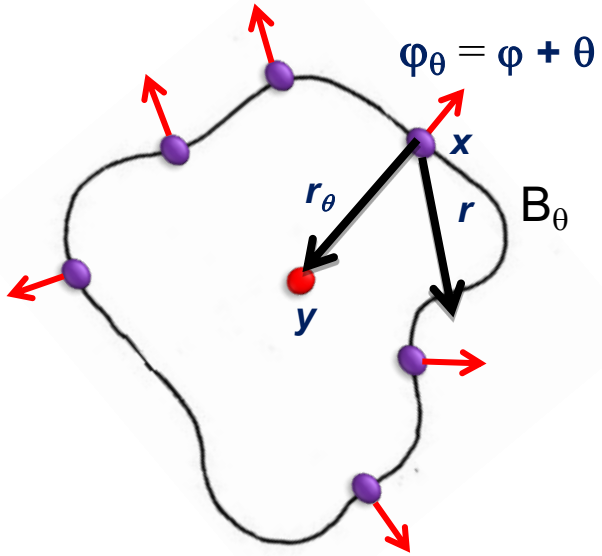


Where is the baricenter?

Generalized Hough Transform

- Can we find the shape if it is rotated?





- We do not know θ, we should quantize rotation and try all of them!

- The same problem arises for scale

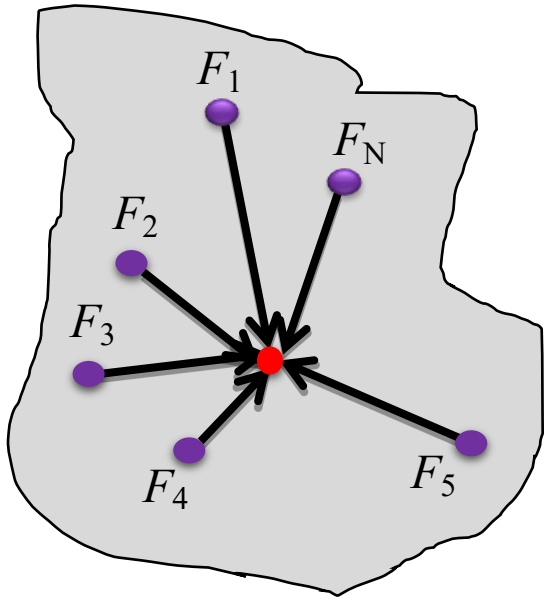
Geometric Validation: Star Model

- Base the GHT feature on local invariant features (e.g., SIFT) and not on edge detection
- DoG features are not found along contour (edge are pruned…) they are found within the object
- Off-line phase:
  - Detect features for each feature points
    - (position, canonical orientation, scale, descriptor)
  - Compute the baricenter and build a star model

$$F = \{F_1, F_2 \dots F_N\},\ F_i = (\mathbf{P}_i, \varphi_i, S_i, \mathbf{D}_i)$$

$$\mathbf{P}_C = \frac{1}{N}\sum_{i=1}^{N} \mathbf{P}_i \rightarrow \mathbf{V}_i = \mathbf{P}_C - \mathbf{P}_i$$

- Add V_i => the joining vector $\forall F_i \in F$ $F_i = (\mathbf{P}_i, \varphi_i, S_i, \mathbf{D}_i, \mathbf{V}_i)$
- No longer use the R table



Geometric Validation

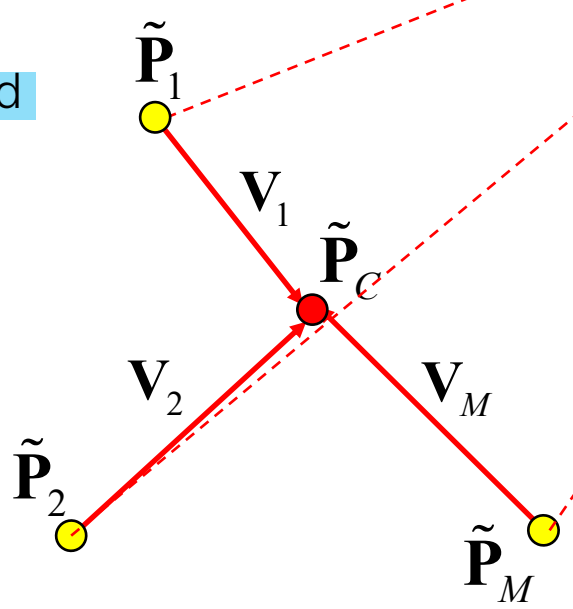
On-line phase

- Extract features from target image
- Match features from the target image to the template
  - Now the matching is based on descriptor not gradient directions

$$\tilde{F} = \left\{ \tilde{F}_1, \tilde{F}_2 \ldots \tilde{F}_M \right\},$$

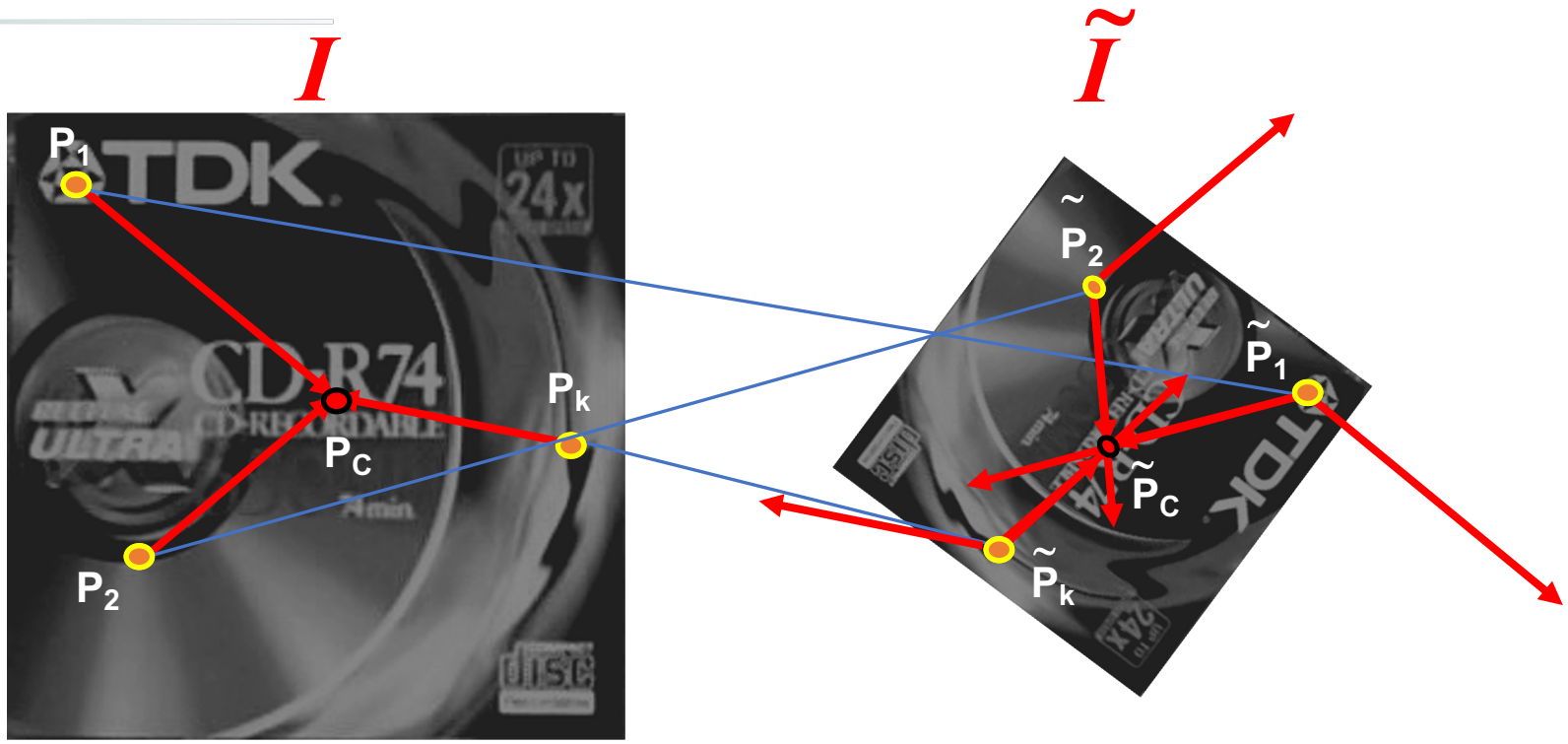
$$\tilde{F}_i = (\tilde{\mathbf{P}}_i, \tilde{\varphi}_i, \tilde{S}_i, \tilde{\mathbf{D}}_i)$$

$$\tilde{D}_i \Leftrightarrow D_i, i = 1 \ldots M$$



Similarity Invariant Voting

- The joining vectors are not pointing coherently towards the reference point
- We should:
  - rotate all of them by the rotation obtained as the difference with the canonical rotation
  - Scale them according to the ratio of scales



$$F_i = (\mathbf{P}_i, \varphi_i, S_i, \mathbf{D}_i, \mathbf{V}_i)$$

$$\tilde{F}_i = (\tilde{\mathbf{P}}_i, \tilde{\varphi}_i, \tilde{S}_i, \tilde{\mathbf{D}}_i)$$

$$\Delta\varphi_i = \tilde{\varphi}_i - \varphi_i$$

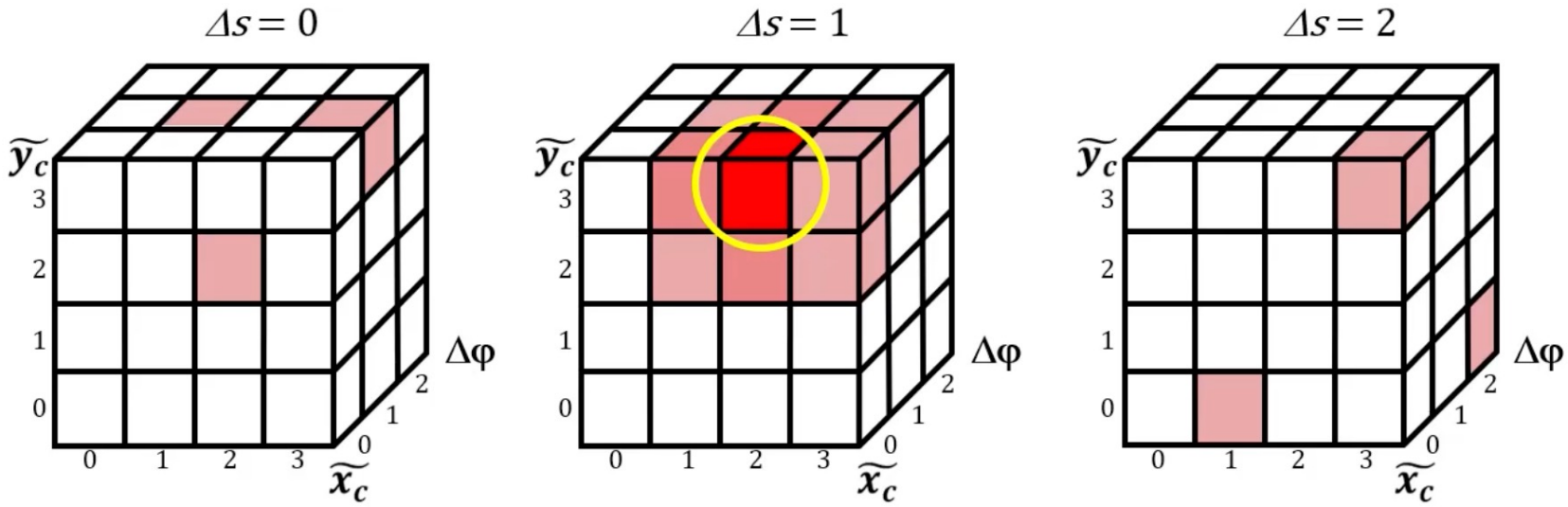
$$s_i = \frac{\tilde{S}_i}{S_i}$$

$$\tilde{\mathbf{P}}_{C_i} = \tilde{\mathbf{P}}_i + s_i \cdot \mathbf{R}(\Delta\varphi_i)\mathbf{V}_i$$

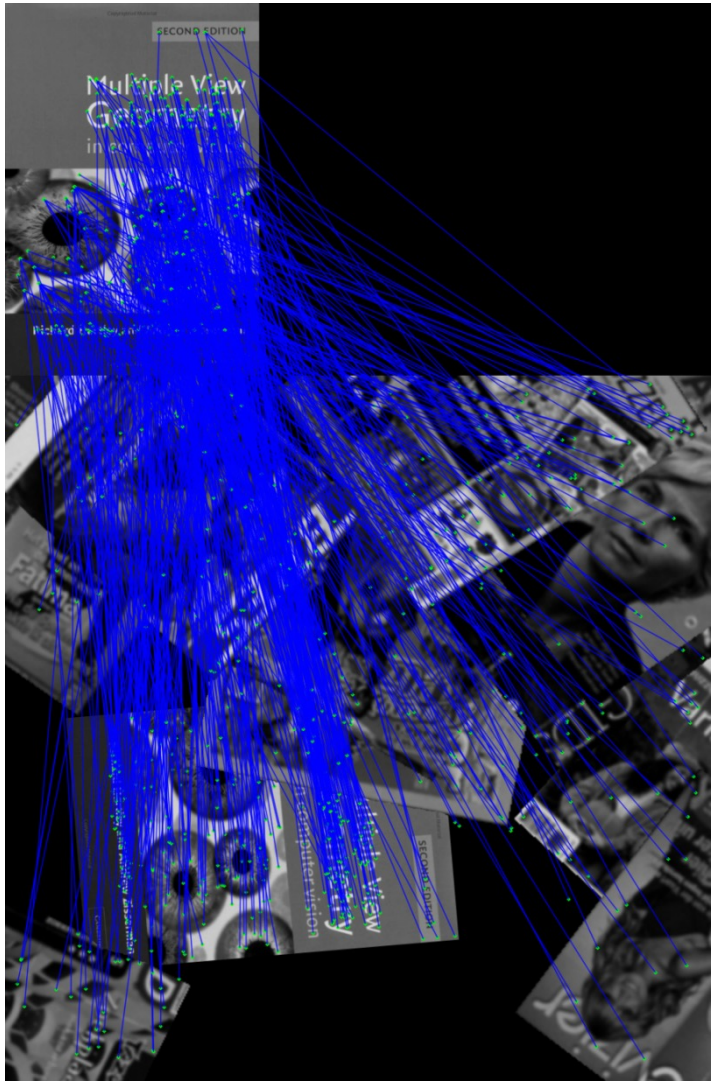
$$\mathbf{A}[\tilde{\mathbf{P}}_{C_i}] + +$$

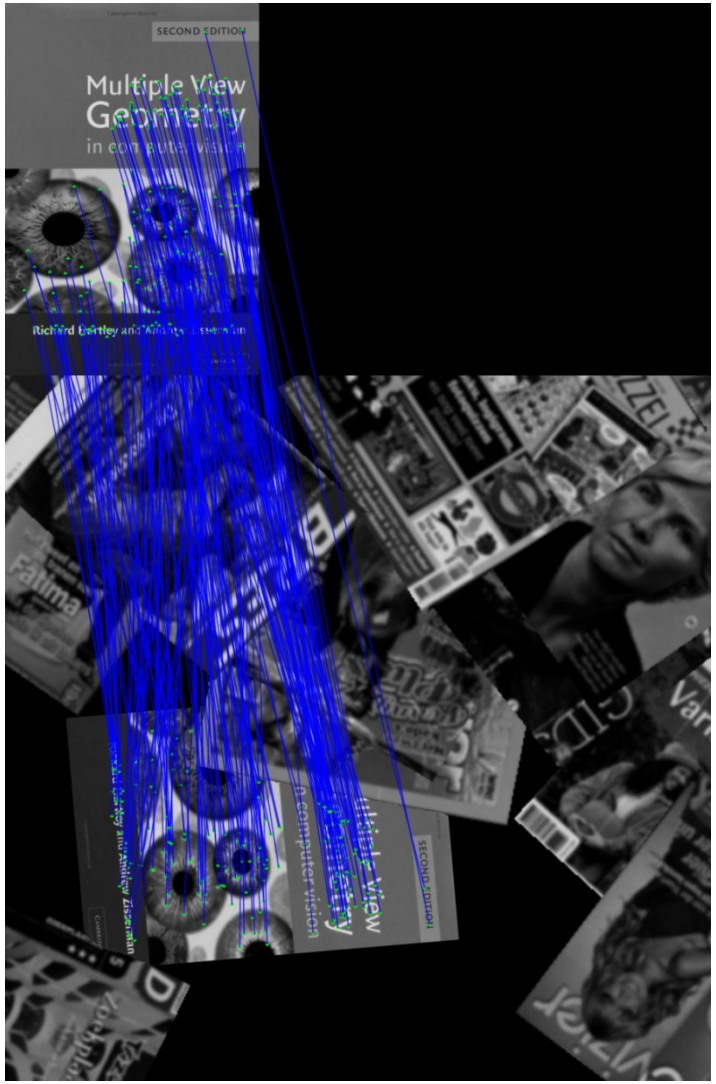
Voting process

- Each match cast a vote for the position of the baricentre
- Use an accumulator array (quantized), increment each translation bin if a prediction hits a specific quantized position (voting process as before).
- The accumulation array is 4 dimensional because we need to accumulate also on the base of the rotation and scale hypothesis, otherwise we accumulate in the same translation bin but for different hypothesis of rotations and scales.



Example


SECOND EDITION
Multiple View
Geometry


SECOND EDITION
Multiple View
Geometry