Matching and Tracking

- Goal: develop matching procedures that can recognize and track objects when
  - objects are partially occluded
  - image cannot be segmented by thresholding
- Key questions:
  - How do we represent the appearance of an object?
  - How do we match these representations against an image?

Solution 1: Image correlation

- Given:
  - n x n image, M, of an object of interest. This is called a **template** and is our representation of the appearance of the object.
  - n x n image, N, that possibly contains that object (usually a window of a larger image)
- Goal: Develop functions that compare M and N and measure the similarity of M and N
  - sum of squared differences: $SSD = \sum_{i=1}^{n}\sum_{j=1}^{n}[M(i,j) - N(i,j)]^2$
  - correlation:

$$C = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} M(i,j)N(i,j)}{[\sum_{i=1}^{n}\sum_{j=1}^{n} M(i,j)^2 \sum_{i=1}^{n}\sum_{j=1}^{n} N(i,j)^2]^{1/2}}$$

Image correlation

- This correlation measure takes on values in the range [0,1]
  - it is 1 if and only if N = cM for some constant c
    - so N can be uniformly brighter or darker than the template, M, and the correlation will still be high.
    - the SSD is sensitive to these differences in overall brightness
  - The first term in the denominator, $\Sigma\Sigma M^2$ depends only on the template, and can be ignored
  - The second term in the denominator, $\Sigma\Sigma N^2$ can be eliminated if we first normalize the grey levels of N so that their total value is the same as that of M - just scale each pixel in N by $\Sigma\Sigma M / \Sigma\Sigma N$
    - practically, this step is sometimes ignored, or M is scaled to have average grey level of the big image from which the unknown images, N, are drawn.

Image correlation

- Suppose that M(i,j) = cN(i,j)

$$C = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} M(i,j)N(i,j)}{[\sum_{i=1}^{n}\sum_{j=1}^{n} M(i,j)^2 \sum_{i=1}^{n}\sum_{j=1}^{n} N(i,j)^2]^{1/2}}$$

$$= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} cN(i,j)N(i,j)}{[\sum_{i=1}^{n}\sum_{j=1}^{n} c^2 N(i,j)^2 \sum_{i=1}^{n}\sum_{j=1}^{n} N(i,j)^2]^{1/2}}$$

$$= \frac{c\sum_{i=1}^{n}\sum_{j=1}^{n} N(i,j)^2}{c[\sum_{i=1}^{n}\sum_{j=1}^{n} N(i,j)^2 \sum_{i=1}^{n}\sum_{j=1}^{n} N(i,j)^2]^{1/2}}$$

$$= 1$$

Image correlation

- Alternatively, we can rescale both M and N to have unit total intensity
  - N’(i,j) = N(i,j)//ΣΣN
  - M’(i,j) = M(i,j)/ΣΣM
- Now, we can view these new images, M’ and N’ as unit vectors of length n^2.
- The correlation measure ΣΣM’(i,j)N’(i,j) is the familiar dot product between the two n^2 vectors **M’** and **N’**
  - recall that the dot product is the cosine of the angle between the two vectors
  - it is equal to 1 when the vectors are the same vector, or the normalized images are identical
- These are BIG vectors

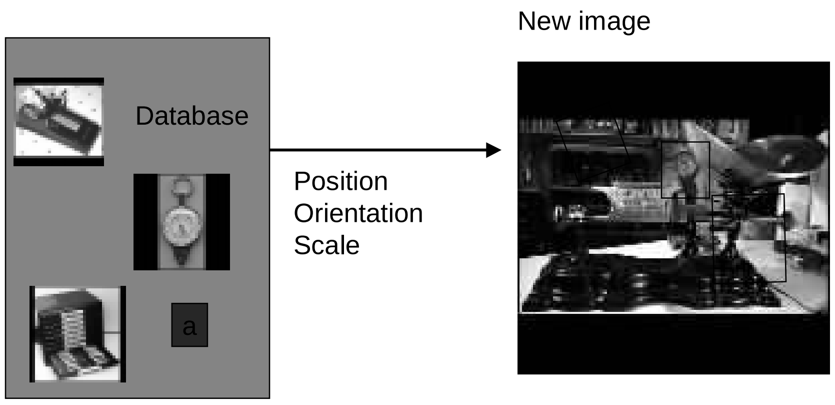
Reducing the computational cost of correlation matching

- A number of factors lead to large costs in correlation matching:
  - the image N is much larger than the template M, so we have to perform correlation matching of M against every nxn window of N
  - we might have **many** templates, M_i, that we have to compare against a given image N
    - face recognition - have a face template for **every** known face; this might easily be tens of thousands
    - character recognition - template for each character
  - we might not know the orientation of the template in the image
    - template might be rotated in the image N - example: someone tilts their head for a photograph
    - would then have to perform correlation of rotated versions of M against N

Reducing the computational cost of correlation matching

- A number of factors lead to large costs in correlation matching:
  - we might not know the scale, or size, of the template in the unknown image
    - the distance of the camera from the object might only be known approximately
  - would then have to perform correlation of scaled versions of M against N

Template matching



Reducing the cost of template matching

- Reducing the number of image windows that need to be compared against the database
  - find “objects” in the image
- Reducing the number of database objects that need to be matched against any window
  - index templates by features such as moments that are not changed by rotations and scale changes
  - measure moments of candidate windows
  - only match “similar” templates
- Reducing the number of operations needed to match a given template to the image

Reducing the computational cost of correlation matching

- Two basic techniques for reducing the number of operations associated with correlation
  - reduce the number of pixels in M and N
    - multi-resolution image representations
    - principal component or “feature selection” reductions
  - match a subset of M against a subset of N
    - random subsets
    - boundary subsets - edge correlation

Multi-resolution correlation

- Multi-resolution template matching
  - reduce resolution of both template and image by creating an **image pyramid**
  - match small template against small image
  - identify locations of strong matches
  - expand the image and template, and match higher resolution template selectively to higher resolution image
  - iterate on higher and higher resolution images
- Issue:
  - how to choose detection thresholds at each level
    - too low will lead to too much cost
    - too high will miss match

Image pyramids

- Base of the pyramid, level 0, is the full resolution image - say 2^n x 2^n
- Level i of the pyramid is obtained from level i-1 as follows
  - partition level i-1 into non-overlapping 2^k x 2^k blocks
    - typically, k = 1 or 2
  - compute an average grey level in each of these blocks
    - unweighted average
    - Gaussian weighted average more typical
  - assign that average grey level to the corresponding level i pixel
- **For you to think about: How many pixels are there in an image pyramid having an nxn base and a reduction by neighborhoods of size 2^k x 2^k?**

Example

8	7	8	12				
13	12	7	6				
30	32	34	30				
32	26	28	33				

10	8		
30	31		

20	

Subset matching techniques

- Subtemplate/template matching
  - choose a subset of the template
  - match it against the image
  - compare the remainder of the template at positions of high match
    - can add pieces of the template iteratively in a multistage approach
- Key issues:
  - what piece(s) to choose?
    - want pieces that are rare in the images against which we will perform correlation matching so that non-match locations are identified quickly
    - choose pieces that define the geometry of the object
  - how to choose detection thresholds at each stage

Subset matching methods - edge correlation

- Reduce both M and N to edge maps
  - binary images containing “1” where edges are present and “0” elsewhere
  - associated with each “1” in the edge map we can associate
    - location (implicitly)
    - orientation from the edge detection process
    - color on the “inside” of the edge for the model,M, and on both sides of the edge for the image, N,

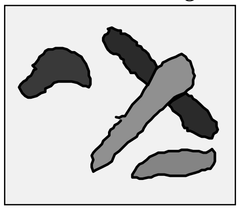
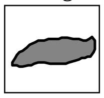
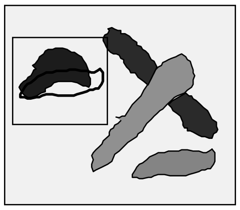


Image N


Template M

Edge template matching

- Simple case
  - N and M are binary images, with 1 at edge points and 0 elsewhere
  - The match of M at position (i,j) of N is obtained by
    - placing M(0,0) at position N(i,j)
    - counting the number of pixels in M that are 1 and are coincident with 1’s in N - binary correlation



$$C(i,j) = \sum_{r=1}^{n}\sum_{s=1}^{n} M(r,s) \times N(r+i, s+j)$$

Observations

- Complexity of matching M against N is O(n^2m^2) for an nxn template and mxm image
  - to allow rotations of M, must match rotated versions of M against N
  - to allow for scale changes in M, must match scaled versions of M against N
- Small distortions in the image can give rise to very bad matches
  - can be overcome by "binary smoothing" (expansion) of either the template or the image
  - but this also reduces the "specificity" of the match

Hough transforms

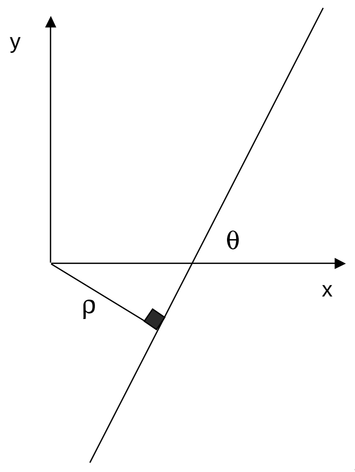
- Consider the following simple problem:
  - Given: a binary image obtained by thresholding and thinning the results of edge detection
  - Find
    - **a)** the largest collinear subset of 1's in that binary image
    - **b)** all collinear subsets of size greater than a threshold t
    - c) a set of disjoint collinear subsets of size greater than a threshold t
- Brute force algorithm
  - For every possible line
    - generate a list of image coordinates on that line and count the number of edge points at those coordinates
- What is the set of all possible lines?

Representing lines

- Parametric representation of lines
  - y = mx + b
    - m is the slope
    - b is the y-intercept
  - problems
    - m is unbounded
    - cannot represent vertical lines

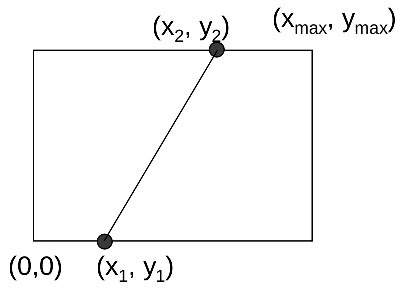
Parametric representation of lines (ρ,θ)

- $\rho = x\cos\theta + y\sin\theta$
- ρ is an unbounded parameter in the representation, but is bounded for any finite image
- θ, the slope parameters, is bounded in the interval $[0,2\pi]$



Parametric representation of lines (x,y,x',y')

- Encode a line by the coordinates of its two intersections with the boundary of the image
- all parameters are bounded by the image size
- but now we have 4 rather than two parameters

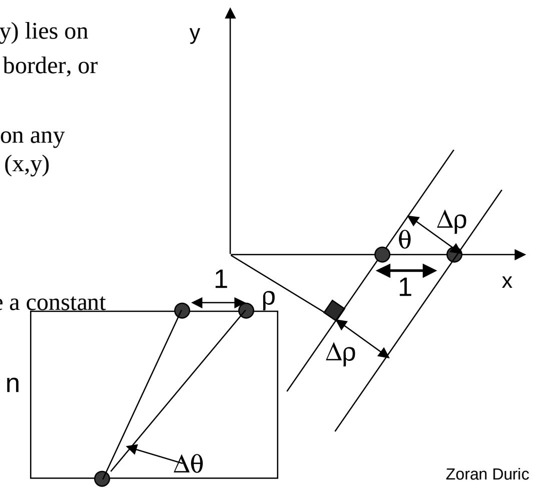


Brute force solution to line detection

- Brute force algorithms enumerates L, the set of "all" lines passing through B.
  - for each line in L it generates the image pixels that lie on that line
  - it counts the number of those pixels in B that are 1's
    - for problem (a) it remembers the maximal count (and associated line parameters) greater than the required threshold
    - for problem (b) it remembers all that satisfy the threshold requirement.
- So, how do we
  - enumerate L
  - given an element, λ, of L enumerate the pixels in B that lie on λ

Brute force solution

- Enumeration of L
  - (x,y,x’,y’) - easy: each (x,y) lies on
    - one side of the image border, or
    - a corner
    - (x’,y’) can be a point on any border not containing (x,y)
  - (ρ,θ) - much harder
    - Δρ = sin θ
    - Δθ ≅ 1/n
    - practically, would use a constant quantization of ρ.



Generating the pixels on a line

- Standard problem in computer graphics
- Compute the intersections of the line with the image boundaries
  - let the intersection be (x_1, y_1), (x_2, y_2)
  - Compute the “standard” slope of the line
    - special cases for near vertical line
  - if the slope is < 1, then then the y coordinate changes more slowly than x, and the algorithm steps through x coordinates, computing y coordinates - depending on slope, might obtain a run of constant y but changing x coordinates
  - if the slope >= 1, then x changes more slowly than y and the algorithm will step through y coordinates, computing x coordinates

Drawbacks of the brute force algorithm

- The complexity of the algorithm is the sum of the lengths of all of the lines in L
  - consider the [(x_1, y_1), (x_2, y_2)] algorithm
  - there are about 3n possible locations for (x_1, y_1) and there are 2n possible locations for (x_2, y_2) once (x_1, y_1) is chosen(this avoids generating lines twice). This is $6n^2$ lines
  - It is hard to compute the average length of a line, but it is O(n).
  - So, the brute force algorithm is $O(n^3)$
- Many of these lines pass through all or almost all 0’s
  - practically, the 1’s in our binary image were generated by an edge or feature detector
  - for typical images, about 3-5% of the pixels lie on edges
  - so most of the work in generating lines is a waste of time.

Hough transform

- Original application was detecting lines in time lapse photographs of bubble chamber experiments
  - elementary particles move along straight lines, collide, and create more particles that move along new straight trajectories
  - Hough was the name of the physicist who invented the method
- Turn the algorithm around and loop on image coordinates rather than line parameters
- Brute force algorithm:
  - For each possible line, generate the line and count the 1’s
- Hough transform
  - For each possible line pixel (x,y) (1 in B) generate the set of all lines passing through (x,y)

Hough transform

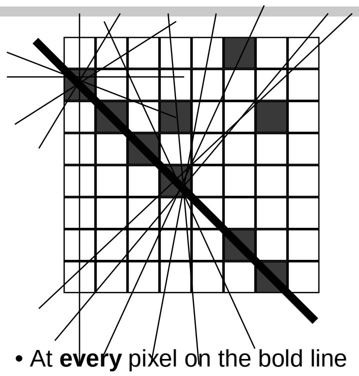
- Algorithm uses an array of accumulators, or counters, H to tally the number of 1’s on any line
  - we will use the (ρ,θ) representation, so each element of H, referenced by $H(\rho,\theta)$, is the counter for the line $\rho = x\cos\theta + y \sin\theta$
  - size of this array is determined by the quantization of the parameters in the chosen line representation
  - when the algorithm is completed, $H(\rho,\theta)$ will contain the number of points from B that satisfy the equation (i.e, lie on the line) $\rho = x\cos\theta + y \sin\theta$
- Algorithm scans B. Whenever it encounters a 1 at a pixel (x,y) it performs the following loop
  - for $\theta = 0, 2\pi, \Delta\theta$

```
    $\rho = x\cos\theta + y \sin\theta$
    $H[\rho norm(\rho), \theta norm(\theta)] = H[\rho norm(\rho), \theta norm(\theta)] + 1$
```

  - norm turns the floats into valid array indices

Hough transform

- What is the computational complexity of the Hough transform?
  - Scanning the image is $O(n^2)$ and if we encounter a fixed percentage of 1’s, we still need to nontrivially process $O(n^2)$ pixels
  - At each pixel, we have to generate $O(n)$ lines that pass through the pixel
  - So it is **also** $O(n^3)$ in the worst case
  - But practically, the Hough transform only does work for those pixels in B that are 1’s
  - This makes it **much** faster than the brute force algorithm



- At **every** pixel on the bold line the Hough transform algorithm will cast a vote for that line
- When the algorithm terminates, that bin will have a score equal to the number of pixels on the line

Solving the original problems

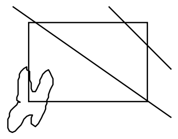
- Problem (a) - Find the k lines having maximal score
  - compute the Hough transform
  - scan array H for the k maximal values; resolve ties arbitrarily
  - **problem:** scanning H can be time consuming.
    - Alternatively, can keep track of the locations in H having maximal tally as the algorithm proceeds using a priority queue
- Problem (b) - Find all lines having score > t
  - compute the Hough array
  - scan the array for all values > t
  - **problem:** also requires scanning the array
    - Can maintain a data structure of above threshold elements of H and add elements to this data structure whenever the algorithm **first** sends an entry of H over t.

Solving the original problems

- Problem (c) - find a set of disjoint lines all of which have size greater than a threshold t.
  - compute the Hough transform, H
  - scan H for the highest value; if it is less then t, halt. If it is >= t, add it to the set (*)
  - remove the “votes” cast by the points on that line
    - use our line generation algorithm to enumerate the image points on that line
    - subtract the votes cast for all elements of H by the 1’s on that line
    - this ensures that a point in the image will contribute to the score for one and only one line as the lines are extracted
  - go back to (*)
- It is difficult to see how to avoid the scanning of H after iteration 1

Other practical problems

- Algorithm is biased towards long lines
  - the number of pixels on the intersection of a line and the image varies with ρ and θ.
  - When we generalize this algorithm to detect other types of shapes, the bias will be introduced by the border of the images **clipping** the shapes for certain placements of the shapes in the image.
  - Can precompute, for each (ρ,θ), the number of pixels on the line $\rho = x\cos\theta + y\sin\theta$ and place these in a normalization array, η, which is exactly the same size as H
  - After the accumulator array is completed, we can divide each entry by the corresponding entry in η to obtain the percentage of pixels on the line that are 1 in B.
  - Similar tricks can be developed to avoid scanning H



Asymptotic complexity

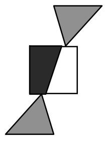
- In the worst case, the Hough transform algorithm is an $O(n^3)$ algorithm, just like the brute force algorithm.
- Consider the following alternative approach
  - generate all pairs of pixels in B that have value 1
    - these define the set of all line segments that will have counts > 1 after running the conventional Hough transform algorithm
  - for each pair, compute the parameters of the line joining that pair of points
    - not necessary to quantize the parameters for this version of the algorithm
  - generate the set of pixels on this line and count the number of 1's in B in this set. This is the number of 1's in B that fall on this line
  - Generate a data structure of all such lines, sorted by count or normalized count. Can be easily used to solve problems (a) and (b)

Asymptotic complexity

- What is the complexity of this algorithm?
  - Again, if there are O(n) 1's in B, then we generate n^2 lines
  - Each of these has O(n) points on it that have to be examined from B
  - So the algorithm is still $O(n^3)$
- Suppose that we **sample** the 1's in B and compute the lines joining only pairs from this sample.
  - If our sample is small - say only the square root of the number of 1's in B, then we will be generating only O(n) lines - one for each pair of points from a set of size $O(n^{1/2})$
  - Incredibly, it can be shown that with **very high probability** any such random sample of size $n^{1/2}$ will contain at least two of the points from any "long" line
  - This method reduces the asymptotic complexity to $O(n^2)$

Using more image information

- Practically, the 1's in B were computed by applying an edge detector to some gray scale image



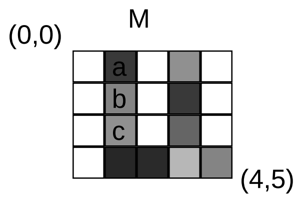
  - this means that we could also associate with each 1 in B the gradient direction measured at that edge point
    - this direction can be used to limit the range of θ considered at each 1 in B - for example, we might only generate lines for θ in the range $[\phi-\pi/4, \phi+\pi/4]$ where ϕ is perpendicular to the gradient direction at a pixel
    - this will further reduce the computational cost of the algorithm
  - each edge also has a gradient magnitude
    - could use this magnitude to differentially weigh votes in the Hough transform algorithm
      - complicates peak finding
      - generally not a good idea - isolated high contrast edges can lead to unwanted peaks

Generalized Hough transform techniques

- Most of the comparisons performed during edge template matching match 0’s in the image N against points in M
  - this is similar to the situation in the brute force line finder, which generates lines containing mostly 0’s in B.
- The Generalized Hough transform avoids comparing the 0’s in the image against the edge template.
  - similar to the Hough transform, the outermost loop of the algorithm will perform computations only when encountering a 1 in N
- Let H(i,j) be an array of counters. This array will be used to accumulate the values of C(i,j)
  - Whenever we encounter a 1 in N we will efficiently determine all placements of M in N that would cause an edge point of T to be aligned with this point of N. These placement will generate indices in C to be incremented

Template representation for the generalized Hough transform

- Rather than represent M as a binary array, we will represent it as a list of coordinates, M’.



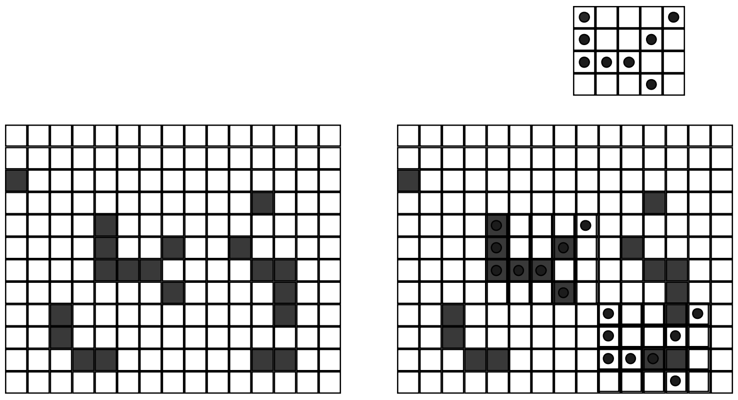
	M’
a	(0,-1)
b	(-1,-1)
c	(-2,-1)
	(-3,-1)
	(-3,-2)
	(-3,-3)
	(-2,-3)
	(-1,-3)
	(0,-3)

- If we place pixel a over location (i,j) in N, then the (0,0) location of the template will be at position (i,j-1)
- If we place pixel c over location (i,j) in N, then the (0,0) location of the template will be at position (i-2,j-1)

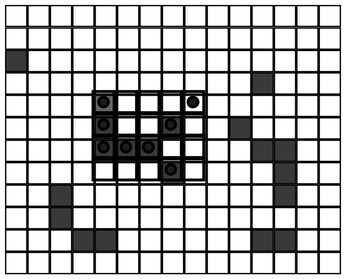
GHT - basic algorithm

- Scan N until a 1 is encountered at position (x,y)
  - Iterate through each element (i,j) from M'
    - The placement of M over N that would have brought M(i,j) over N(x,y) is the one for which the origin of M is placed at position (x+i, y+j).
    - Therefore, we increment H(x+i, y+j) by 1
  - And move on to the next element of M'
- And move on to the next 1 in N
- When the algorithm completes H(i,j) counts the number of template points that would overlay a "1" in N if the template were placed at position (i,j) in N.

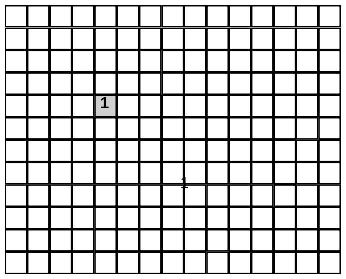
Example



Example

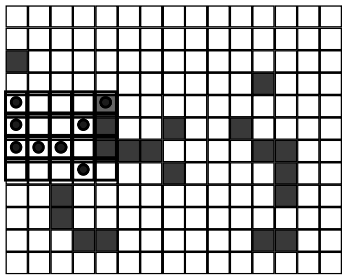


Image

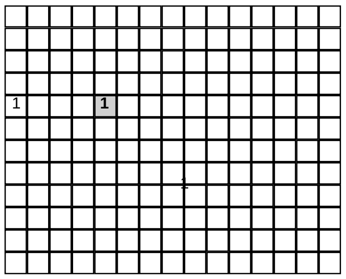


Hough array

Example

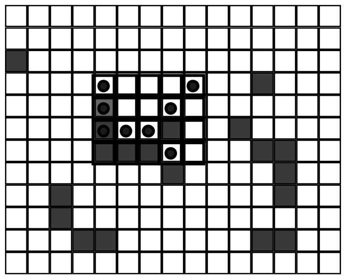


Image

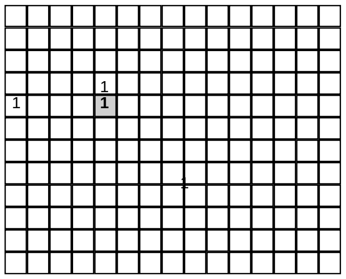


Hough array

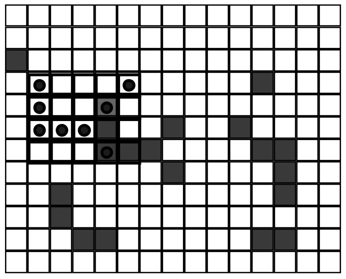
Example



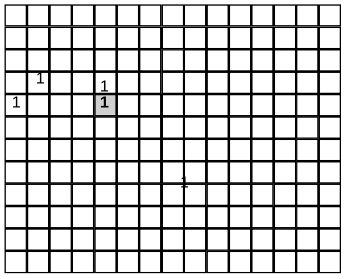
Image


Hough array

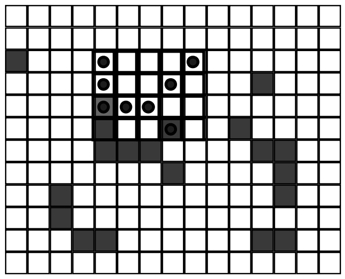
Example



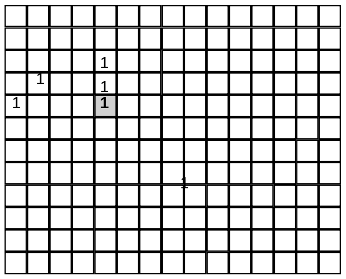
Image


Hough array

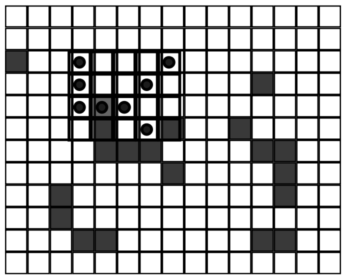
Example



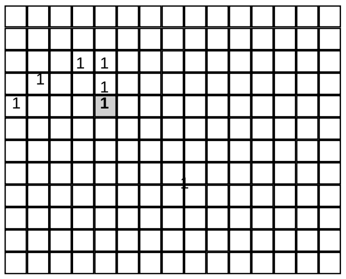
Image


Hough array

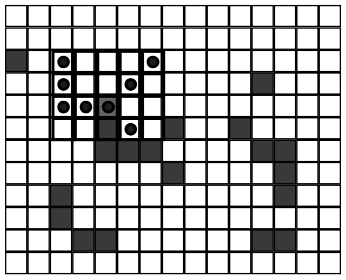
Example



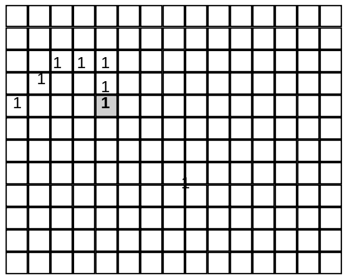
Image


Hough array

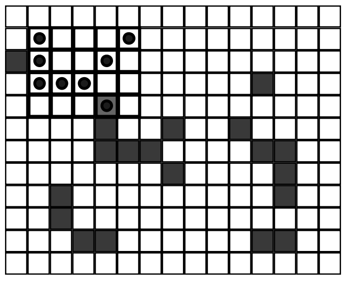
Example



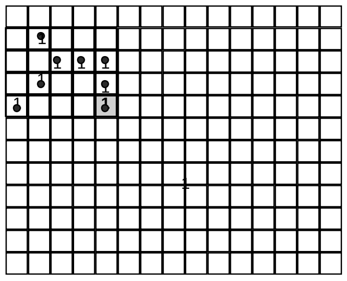
Image


Hough array

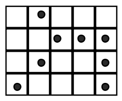
Example



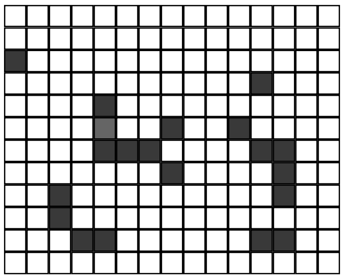
Image


Hough array

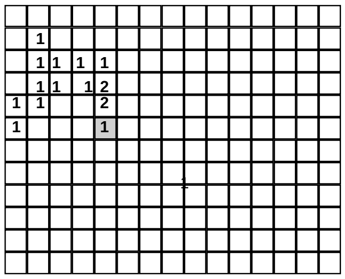
Template


Increment pattern

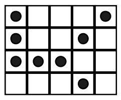
Example

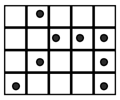


Image

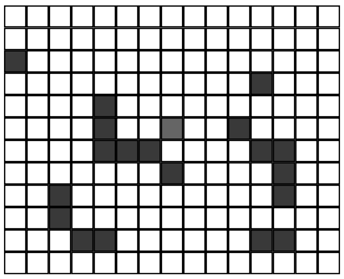


Hough array

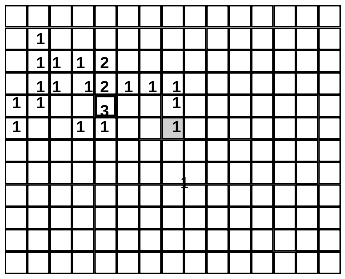

Template


Increment pattern

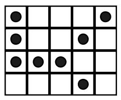
Example

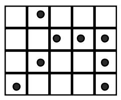


Image

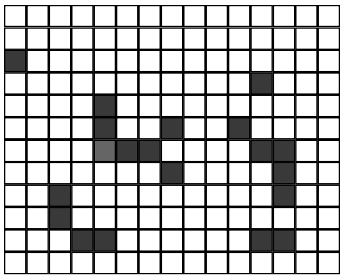


Hough array

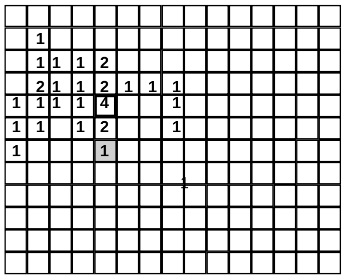

Template

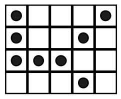

Increment pattern

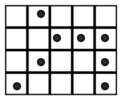
Example



Image


Hough array


Template


Increment pattern

GHT - generalizations

- Suppose we want to detect instances of M that vary in orientation in the image
  - need to increase the dimensionality of H by adding a dimension, θ, for orientation
  - Now, each time we encounter a 1 during the scan of N we must consider all possible rotations of M wrt N - will result in incrementing one counter in each θ plane of H for each point in M.
    - For each (i,j) from M
      - For each quantized θ
      - Determine the placement (r,s) of the rotated template in N that would bring (i,j) onto (x,y) and increment H(r,s,θ)
- For scale we would have to add one more dimension to H and another loop that considers possible scales of M.

Representing high dimensional Hough arrays

- Problems with high dimensional arrays:
  - storage
  - initialization and searching for high values after algorithm
- Possible solutions
  - Hierarchical representations
    - first match using a coarse resolution Hough array
    - then selectively expand parts of the array having high matches
  - Projections
    - Instead of having one high dimensional array store a few two dimensional projections with common coordinates (e.g., store (x,y), (y,θ), (θ,s) and (s,x))
    - Finds consistent peaks in these lower dimensional arrays

GHT generate and test

- Peaks in Hough array do not reflect spatial distribution of points underlying match
  - typical to "test" the quality of peak by explicitly matching template against image at the peak
- Controlling the generate and test framework
  - construct the complete Hough array, find peaks, and test them
  - test as soon as a point in the Hough space passes a threshold
    - if the match succeeds, points in I that matched can be eliminated from further testing
  - test as soon as a point in the Hough space is incremented **even once**

Chamfer matching

- Given:
  - binary image, B, of edge and local feature locations
  - binary "edge template", T, of shape we want to match
- Let D be an array in registration with B such that D(i,j) is the distance to the nearest "1" in B.
  - this array is called the **distance transform** of B
- Goal: Find placement of T in D that minimizes the sum, M, of the distance transform multiplied by the pixel values in T
  - if T is an exact match to B at location (i,j) then M(i,j) = 0
  - but if the edges in B are slightly displaced from their ideal locations in T, we still get a good match using the distance transform technique

Computing the distance transform

- Brute force, exact algorithm, is to scan B and find, for each "0", its closest "1" using the Euclidean distance.
  - expensive in time, and difficult to implement

Computing the distance transform

- Two pass sequential algorithm
- Initialize: set D(i,j) = 0 where B(i,j) = 1, else set D(i,j) = ∞
- Forward pass
  - D(i,j) = min[D(i-1,j-1) + 4, D(i-1,j) + 3, D(i-1, j+1) +4, D(i,j-1) + 3, D(i,j)]
- Backward pass
  - D(i,j) = min[D(i,j+1) + 3, D(i+1,j-1) + 4, D(i+1, j) +3, D(i+1,j+1) + 4, D(i,j)]

Distance transform example

■							
				■			
				■			■
				■	■	■	
							■

0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	2	3	0	1	2	3
3	3	3	1	0	1	2	0
4	4	2	1	0	0	0	1
5	3	2	1	1	1	1	0

f	f	f
f		b
b	b	b

Distance transform example

0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	2	3	0	1	2	3
3	3	3	1	0	1	2	0
4	4	2	1	0	0	0	1
5	3	2	1	1	1	1	0

2	2	3	4	4	4	4	4
1	2	2	3	3	3	3	3
0	1	2	2	2	2	2	3
1	1	2	1	1	1	2	2
2	2	2	1	0	1	1	1
3	3	2	1	0	1	1	0
4	3	2	1	0	0	0	1
4	3	2	1	1	1	1	0

D(i,j) = min[D(i,j), D(i,j+1)+1, D(I+1, j-1)+1, D(i+1,j)+1, D(i+1,j+1)+1]

f	f	f
f		b
b	b	b

Chamfer matching

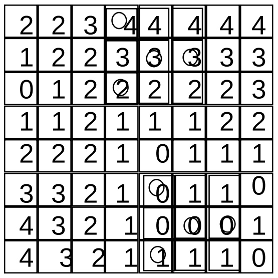
- Chamfer matching is convolution of a binary edge template with the distance transform
  - for any placement of the template over the image, it sums up the distance transform values for all pixels that are “1’s” (edges) in the template
  - if, at some position in the image, all of the edges in the template coincide with edges in the image (which are the points at which the distance transform is zero), then we have a perfect match with a match score of 0.

Example

2	2	3	4	4	4	4	4
1	2	2	3	3	3	3	3
0	1	2	2	2	2	2	3
1	1	2	1	1	1	2	2
2	2	2	1	0	1	1	1
3	3	2	1	0	1	1	0
4	3	2	1	0	0	0	1
4	3	2	1	1	1	1	0

○		
	○	○
○		

Template



Match score is $$\sum_{k=1}^{n}\sum_{l=1}^{n} T(i,j)D(i+k,j+l)$$

Hausdorff distance matching

- Let M be an nxn binary template and N an nxn binary image we want to compare to that template
- H(M,N)) = max(h(M, N), h(N, M)) where
  $$h(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\|$$
  - || || is a distance function like the Euclidean distance function
- h(A,B) is called the directed Hausdorff distance.
  - ranks each point in A based on closeness to a point in B
  - most mis-matched point is measure of match
  - if h(A,B) = e, then all points in A must be within distance e of B.
  - generally, h(A,B) <> h(B,A)
  - easy to compute Hausdorff distances from distance transform

Summary of 2-D object recognition

- Binary vision systems
  - segmentation by thresholding and connected component analysis
  - object modeling using statistical techniques
    - means and variances of global object features such as area, perimeter, etc.
  - recognition using statistical recognition techniques
    - k-nearest neighbors
    - Bayesian recognition
- Drawbacks
  - touching objects
  - occluded objects
  - weak segmentation techniques
- Grey level vision systems
  - (optional) segmentation by edge detection
  - object modeling by templates
    - gray level region templates
    - edge templates (binary)
  - recognition using correlation
    - brute force image correlation
      - speedup methods
    - Hough transform methods
    - Chamfer matching
- Drawbacks
  - computational complexity
    - to support rotations and scaling of templates

Other generalizations

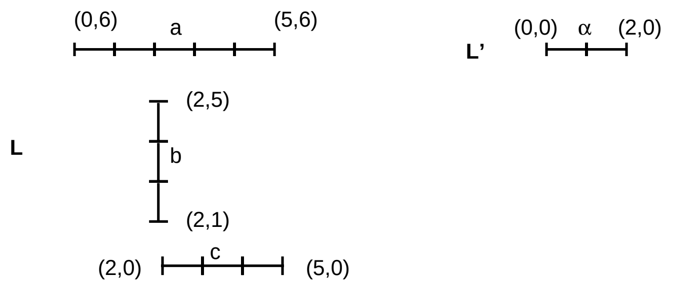
- Match patterns of linear and curvilinear features against images from which such features have been detected
- Impose a hierarchical structure on M, and match pieces and compositions of pieces.
  - at lowest level one finds possible matches to small pieces of M
  - a second GHT algorithm can now find combinations of pieces that satisfy other spatial constraints.

Hough Transforms for line matching

- Let L = {L_1, ..., L_n} be the set of line segments which define M
- Let L’ = {L'_1, ..., L'_m} be the set of observed line segments from N
- Define L_i - L_j as follows:
  - If L_j is a **subsegment** of L_i, then $L_i - L_j = l_j$, where l_j is the length of L_j
  - otherwise $L_i - L_j = 0$
- Let F be a set of transformations that map lines to lines
- Given F, L and L’, find f in F that maximizes:

$$v(f) = \sum_{L_i \in L} \sum_{L_j \in L'} [L_i - f(L_j)]$$

Example - translation only



- Which translations get incremented
  - α-a: (0,6), (1,6), (2,6), (3,6) incremented by 2
  - α-b: none
  - α-c: (2,0), (2,1) incremented by 2

Fast template matching

- Simulated annealing approach
  - Let $T_{\theta,s}$ be a rotated and scaled version of T
  - For a random θ and s, and a random (i,j) match $T_{\theta,s}$ at position (i,j) of I
    - Now, randomly perturb θ,s,i and j by perturbations whose magnitudes will be reduced in subsequent iterations of the algorithm to obtain θ', s', i', j'.
    - Match $T_{\theta',s'}$ at position (i',j'). If the match is better, "move" to that position in the search space. If the match is worse, move with some probability to that position **anyway!**
    - Iterate using smaller perturbations, and smaller probabilities of moving to worse locations
      - the rate at which the probability of taking "bad" moves decreases is called the "cooling schedule" of the process.
  - This has also been demonstrated with deformation parameters that mimic projection effects for planar patterns.