Computer Vision 08 – Image processing and analysis 1c

doc. dr. Janez Perš
(with contributions by prof. Stanislav Kovačič)

Laboratory for Machine Intelligence
Faculty of Electrical Engineering
University of Ljubljana

Quick recap of the previous lectures

- Image formation
- Color
- Image processing
  - Value of the pixel depends only on that pixel
- Local image operations
  - Value of the pixel depends on the small neighborhood of each pixel
  - Examples: convolution (filtering), edge detection
- Edges
  - Smoothing by Gaussian + differentiation in one operation = convolution by the derivative of Gaussian filter

Outline

- Edge detection
  - Canny edge detection
- Corner detection
- Hough transform
- Multi-scale approaches and scale-space

Edge detection

- General approach:
  - Low-pass filtering with Gaussian of suitable sigma to suppress noise
  - Computing derivatives (intensity gradient), applying edge operator
  - Thresholding, thinning if needed
- But, we can combine Gaussian filtering with differentiation.
  - Thus, we compute first derivatives of Gaussian, and filter (convolve) image…
  - … with the derivatives of Gaussian.

Edge detection

- Another class of edge detectors, based on *second order* derivatives,
  - Where *zero crossings* are edge points positions!
  - Again, finite difference method can be used (based on Taylor series approximation of derivatives)
  - Second order central difference approach:

$$f(x+h,y) = f(x,y) + h \times f_x(x,y) + {h^2}/{2} \times f_{xx}(x,y) + O(h^3)$$

$$f(x-h,y) = f(x,y) - h \times f_x(x,y) + {h^2}/{2} \times f_{xx}(x,y) - O(h^3)$$

$$f(x+h,y) + f(x-h,y) = 2 \times f(x,y) + h^2 \times f_{xx}(x,y) - O(h^4)$$

$$f_{xx}(x,y) = \frac{f(x+h,y) - 2 \times f(x,y) + f(x-h,y)}{h^2} + O(h^2)$$

Edge detection

– Second order central difference approach, now for second order derivative of in y direction:

$$f(x, y+h) = f(x, y) + h \times f_y(x, y) + h^2 / 2 \times f_{yy}(x, y) + O(h^3)$$

$$f(x, y-h) = f(x, y) - h \times f_y(x, y) + h^2 / 2 \times f_{yy}(x, y) - O(h^3)$$

$$f(x, y+h) + f(x, y-h) = 2 \times f(x, y) + h^2 \times f_{yy}(x, y) - O(h^4)$$

$$f_{yy}(x, y) = \frac{f(x, y+h) - 2 \times f(x, y) + f(x, y-h)}{h^2} + O(h^2)$$

– Now we can make a convolution with a kernel that produces second derivatives as derived above…

– … and detect edges by observing zero crossings.

Edge detection

– Setting h to 1, and combining the two 2-nd order derivatives, (Laplace operator):

$$\Delta f(x,y) \approx f(x+1,y) + f(x,y+1) - 4\times f(x,y) + f(x,-1,y) + f(x,y-1)$$

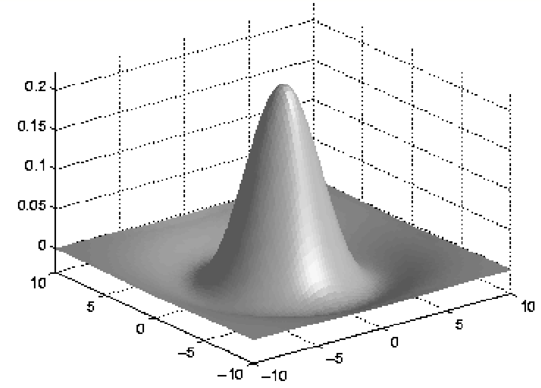
	1	
1	-4	1
	1	

1	1	1
1	-8	1
1	1	1

An equivalent (preferable) version of discrete Laplacian operator

Edge detection

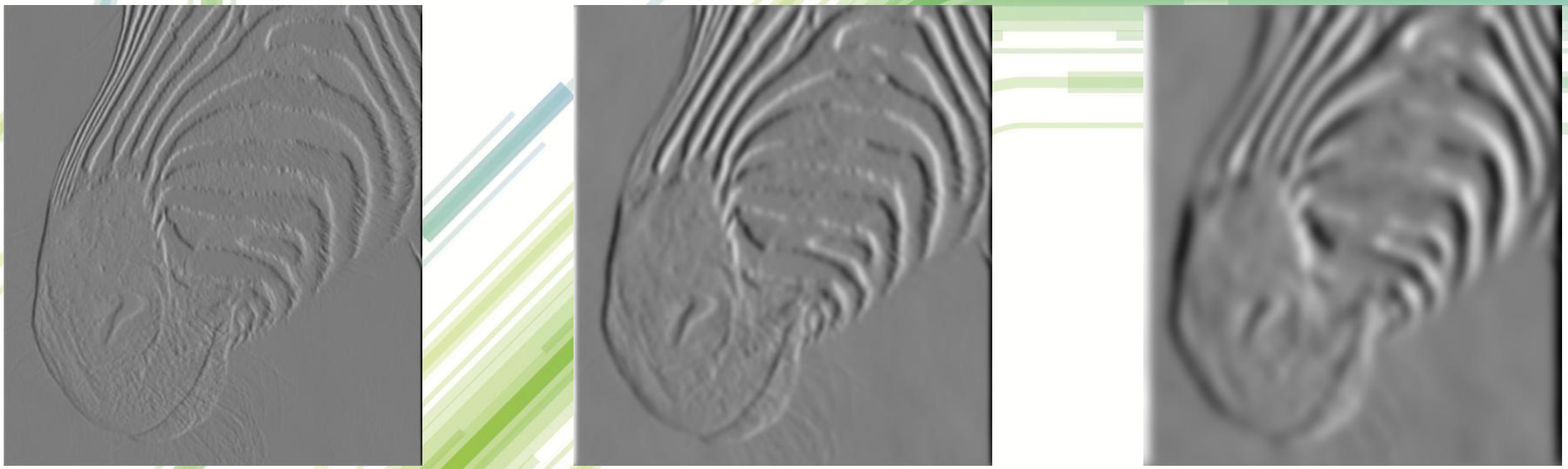
- But second order derivatives are even *more* susceptible to noise!
  - We have to smooth the image first, using the Gaussian filter.
  - Another option is to take second derivatives of Gaussian, combine them into Laplacian, to produce *Laplacian of Gaussian*, or *LoG* for short.



Matlab function: e = edge(I, ‚log‘, ...);

  - Sometimes it is more convenient to take two Gaussians of different sigma, and subtract them, to produce close approximation of LoG,
    DoG – Difference of Gaussians!

Tradeoff between smoothing and localization



Sigma = 1 pixel 3 pixels 7 pixels

- Smoothing removes noise, therefore spurious (false) edge points are largery eliminated. That *improves detection.*
- Nevertheless, smoothing also blurs edges. (thicker ridges)
- Where exactly are then (true) positions of edges?
- Thus, smooting *worsens localization.* This is a tradeoff!

Designing an edge detector

- Criteria for a good – ideal - edge detector:
  - Good detection:
    - the optimal detector should find all real edges, ignoring noise or other artifacts
  - Good localization
    - the edges detected must be as close as possible to the true positions of edges (idealy, at exact position)
  - the detector must return one point only for each true edge point (single response condition)
- In reality, however,
  - there will be some false edges detected, positions misplaced, and more responses to a single edge.

Canny edge detector

- Criteria (J. Canny, 1983):
  - resistance to additive noise in case of a step edge (function)
  - Good localization, response where the actual edge is
  - Single response to an edge

- This is probably the most widely used edge detector in computer vision
  - J. Canny, *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

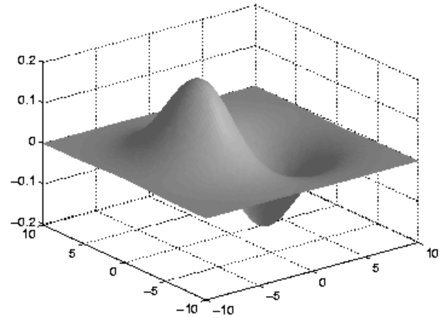
Canny edge detector

- Procedure (implementation)
  - Filtering with Gaussian of suitable size (sigma)
  - Computing derivatives in x and y direction (gradient components)
  - Computing gradient magnitude and gradient direction
  - Non-Maxima Suppression - NMS.
  - Hysteresis thresholding with high and low thresholds

Canny edge detector

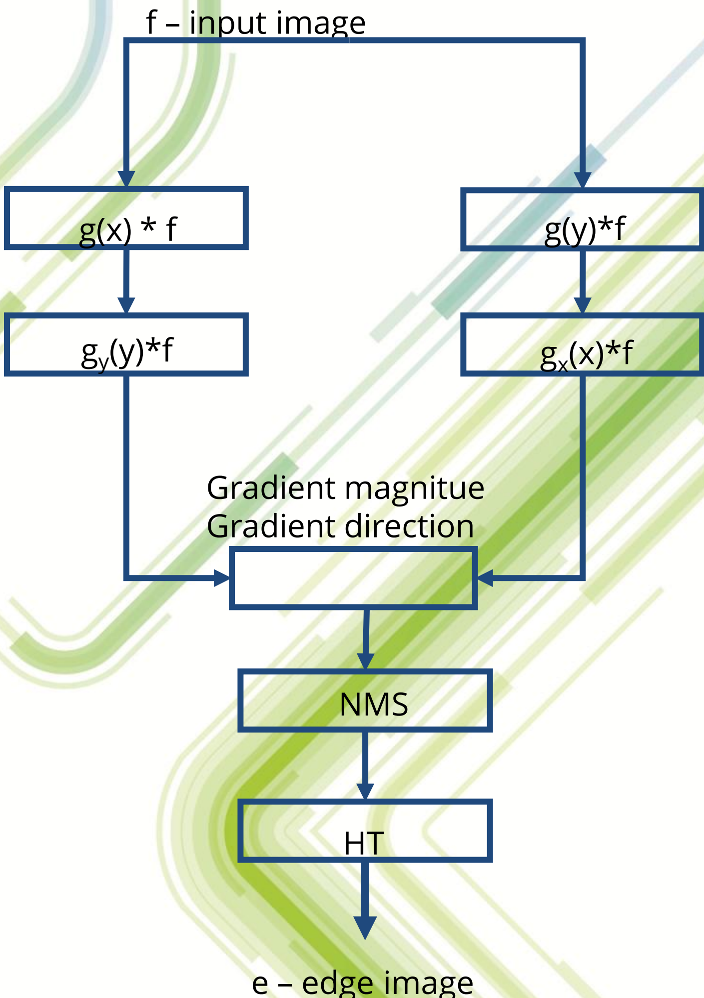
- We take derivatives of filter instead of calculating the derivative of the image.
- Convolve with gaussian derivatives
- Take advantage of kernel separability

$$e(x,y) = \nabla(g(x,y) * f(x,y)) = \begin{bmatrix} \nabla_x g(x,y) * f(x,y) \\ \nabla_y g(x,y) * f(x,y) \end{bmatrix} =$$



$$= \begin{bmatrix} g_x(x,y) * f(x,y) \\ g_y(x,y) * f(x,y) \end{bmatrix} = \begin{bmatrix} g_x(x) * g(y) * f(x,y) \\ g(x) * g_y(y) * f(x,y) \end{bmatrix}$$

Logic of the canny edge detector



- Convolving with Gaussian in x and y
- Convolving with derivatives of Gaussian in y and x direction
- Computing gradient magnitude $||g||^2 = f^2_x + f^2_y$
- Computing gradient direction $\Theta = \text{atan}(f_y / f_x)$
- NMS: Non maxima suppresion
- HT: Hysteresis thresholding

Matlab: *edge(I,…'canny')*

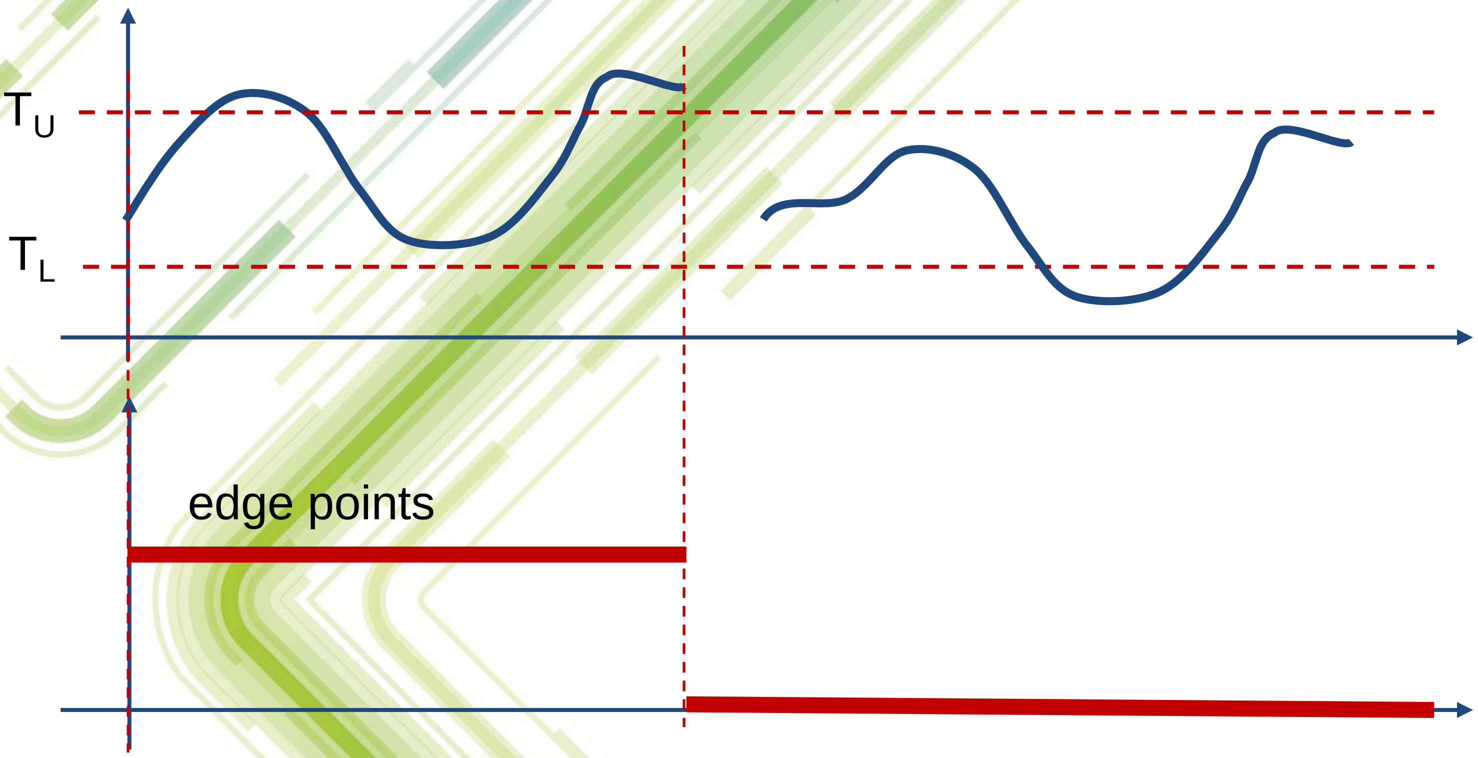
Canny NMS and HT

- Analogy: walking along a ridge
  - With height corresponding to edge image intensity

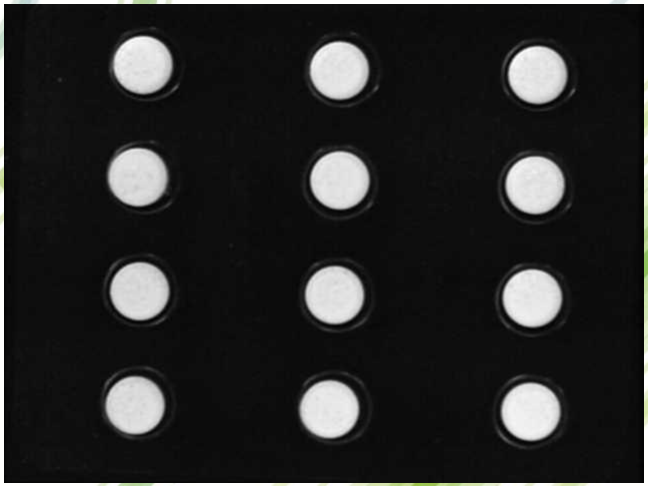
Image source: JPL. IEEE Computer, sep'14, Mojave Crater, NASA JPL (NASA Mars Reconnaissance Orbiter)

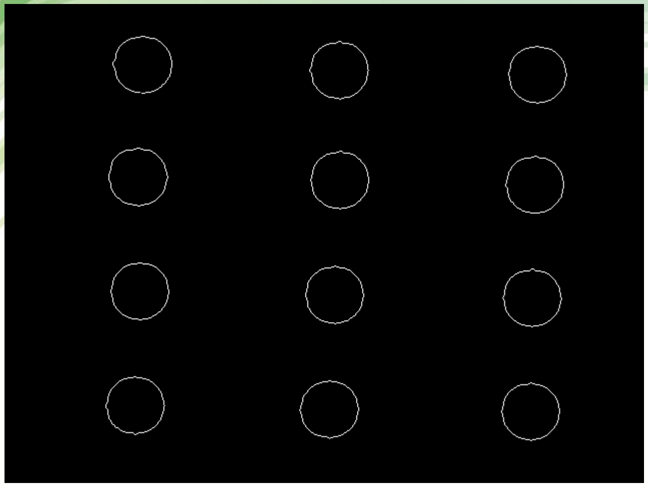
Canny HT

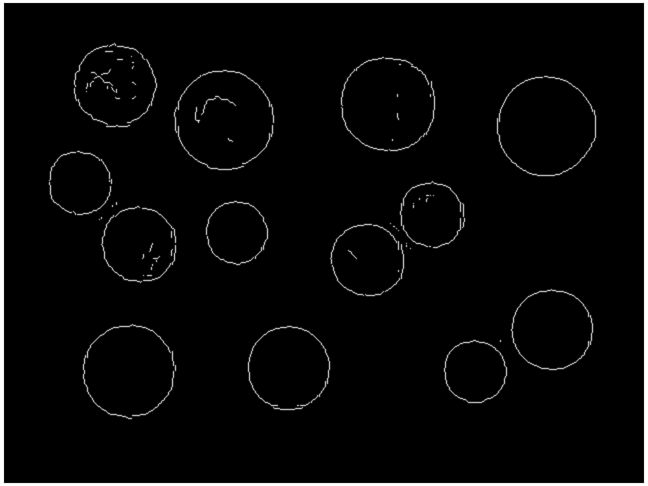
– if gradient mag > T_U pixel is edge, start following
– if gradient mag < T_L pixel is not edge, stop following



Example: Canny detector on tablet and coin images







Example: Sobel vs. Canny on Lena

Original image

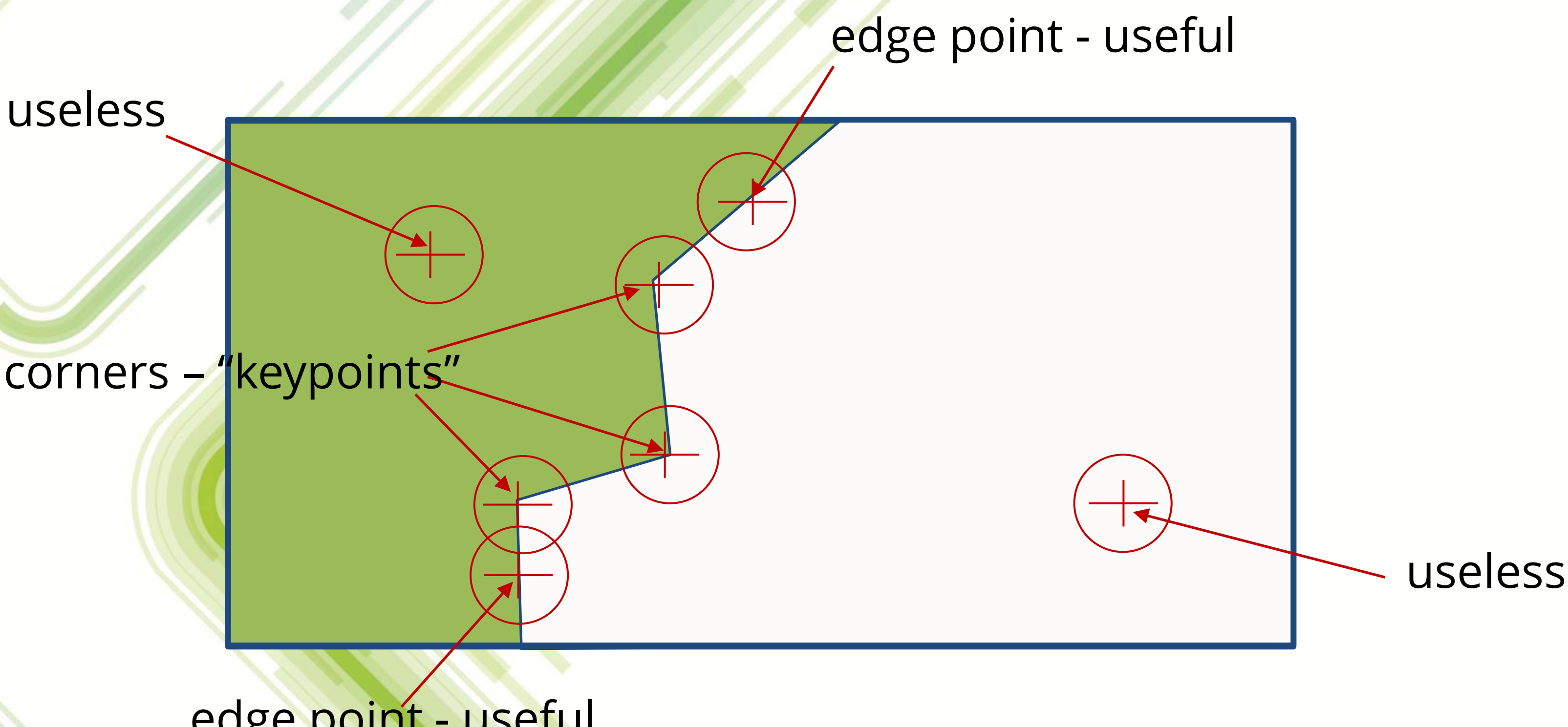
Sobel

Canny

Questions?

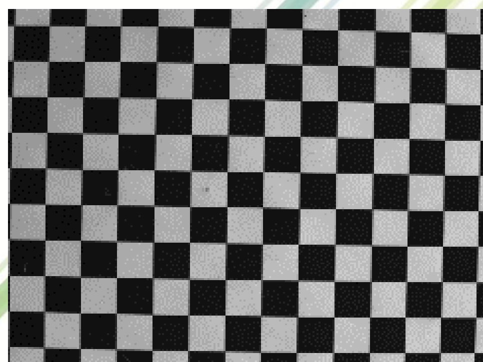
Corner detection

- What is a corner (corner point)?
  - "Corners" are points that differ from their surroundings, preferably in all directions.

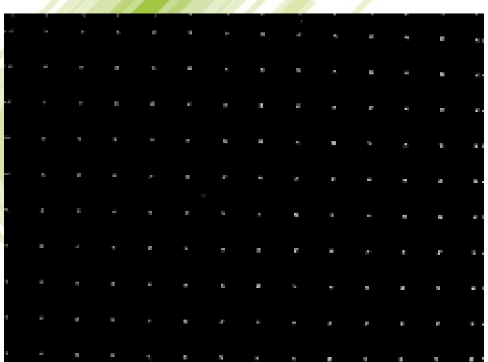


Corner detection

- What are we looking for?
  - We have, e.g. this input



  - And would like to obtain this output (corner points visible):



Corner detection

- Procedure
  - Compute derivatives (e) in x and y direction
  - Compute C in a small neighborhood of (x,y):

$$C(x,y) = \begin{bmatrix} \sum e_x^2 & \sum e_x e_y \\ \sum e_x e_y & \sum e_y^2 \end{bmatrix}$$

- We have C for each (x,y)
  - Compute eigenvalues of C
  - Why?

Corner detection

$$C(x,y) = M \;\Lambda M^T = M \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} M^T$$

- C is symmetric, M is orthogonal
- We declare the point (x,y) "corner" if both eigenvalues are sufficiently large.
- Why?

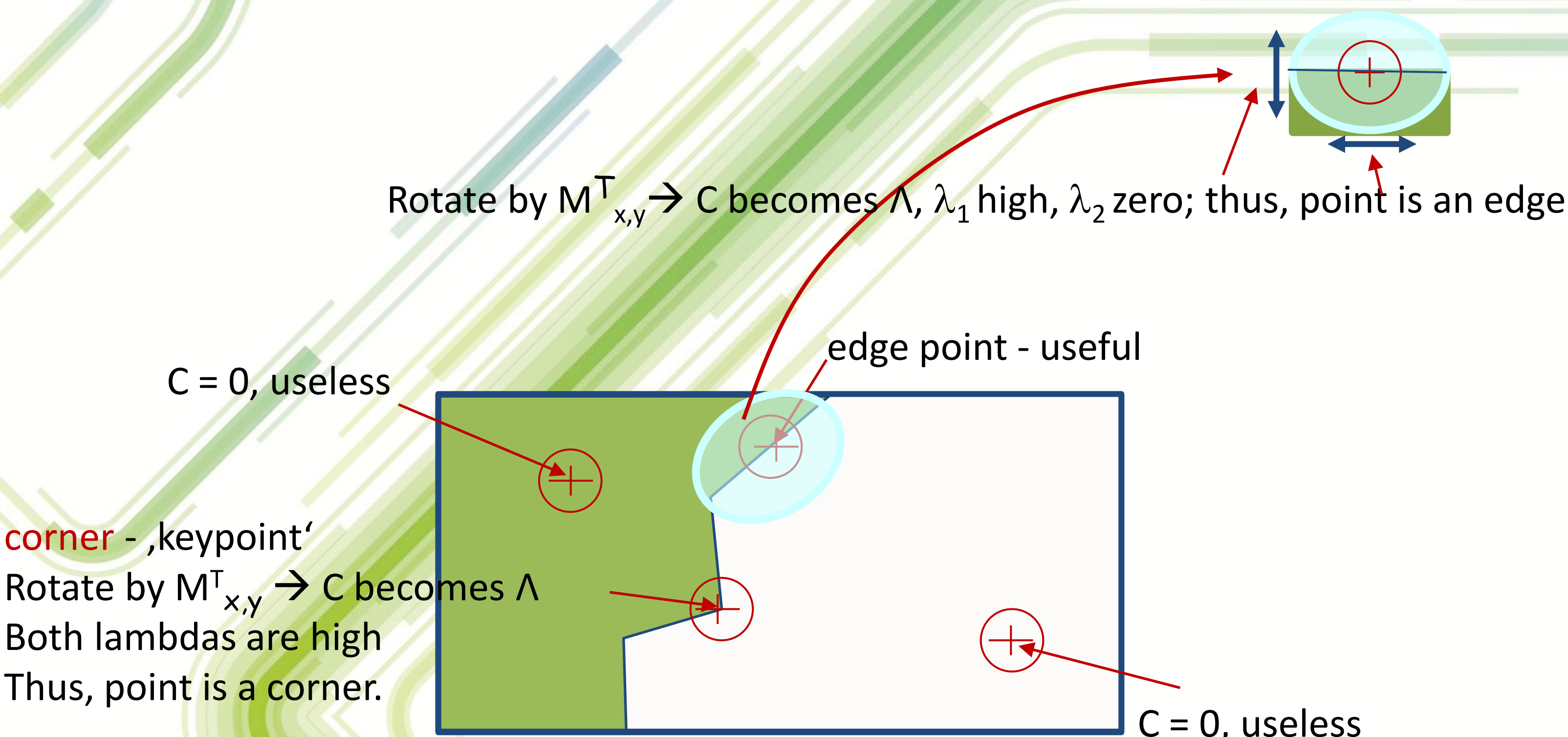
$$C(x,y) = M \;\Lambda M^T$$

- M is orthogonal, i.e. rotation

$$M^T C(x,y) M = \Lambda$$

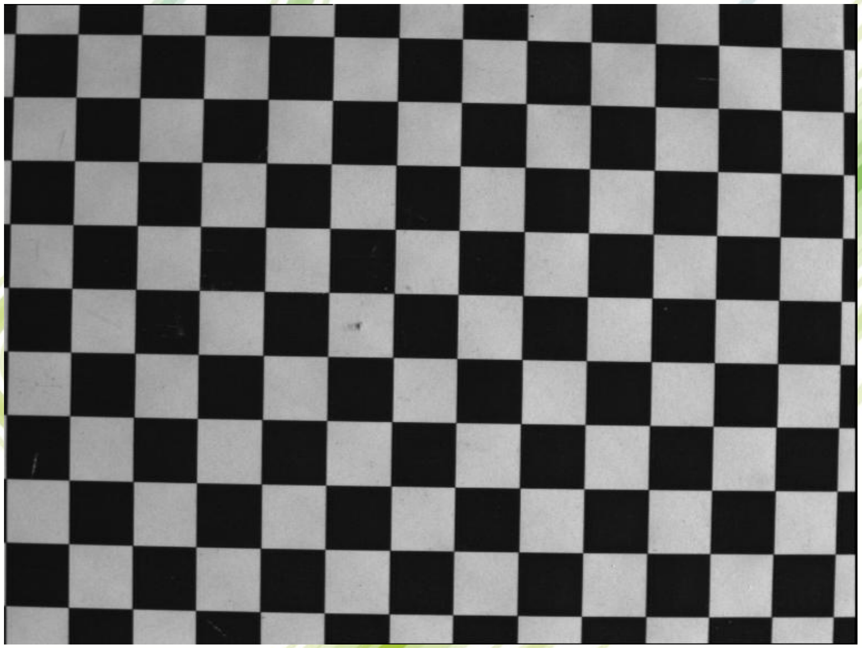
- If coordinate system was rotated by M, C would become diagonal!

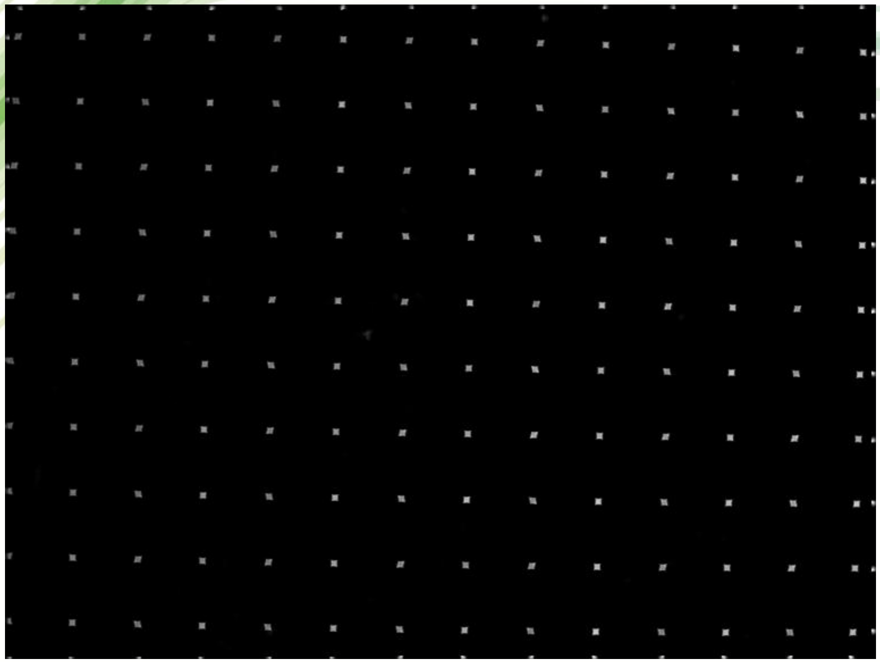
Corner detection



For corners, both λ_1 and λ_2 are large!

Example: corner detection





Matlab: *c=corner(I);*

Example: corner detection on Lena

Generated by P. Corke RVC toolbox function *icorner()*
Only 20 strongest detected corners are shown

Questions?

Hough transform

- So far, we can detect
  - Edges
  - Corners
- Both of those are relatively primitive structures
  - For corner, we get a small “clump” of pixels, where the detector returned high values
  - For an edge, we get a continuous set of pixels for each edge.
- What about high level descriptions/structures?
  - Lines and circles from edges, for example?
  - The answer is Hugh transform!

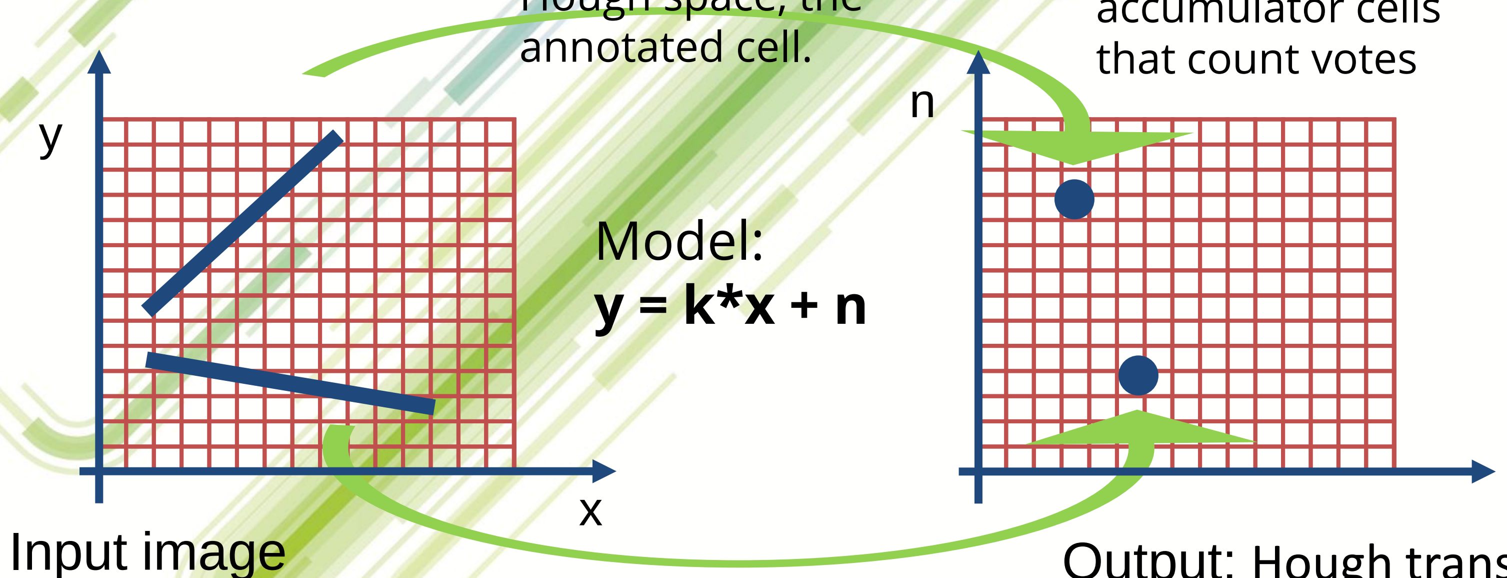
Hough transform

- Hough transform (HT) has been devised by Mr. Hough in sixties.
  - It has been developed and patented for detection of straight line structures.
  - Later on it has been generalized for detection of arbitrary structures.
- The key idea of HT is VOTING. HT is based on a voting principle.
  - Those structures that get more votes are represented more strongly in the image.

Hough transform

All pixels along that line give vote for the same (k,n) cell in Hough space, the annotated cell.

Hough space (HS) is an array of cells – accumulator cells that count votes

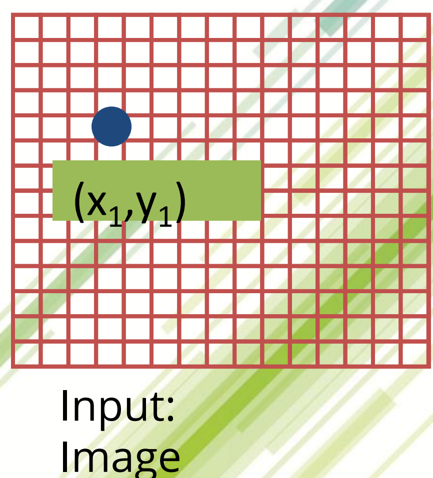


Output: Hough transform for lines.

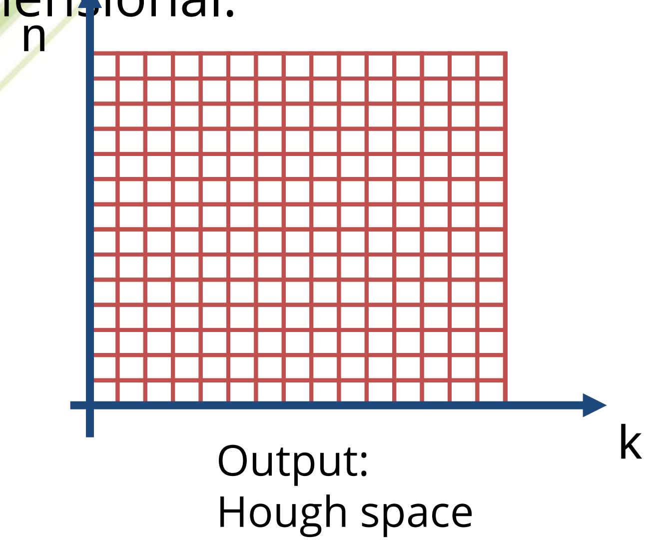
All pixels along that line give vote for another (k,n) cell in Hough space, the annotated cell.

Hough transform for lines

- Hough Space (HS) is a parametric space.
- In our case the model is a line equation with two parameters, *k* and *n*. Therefore, HS is two-dimensional.



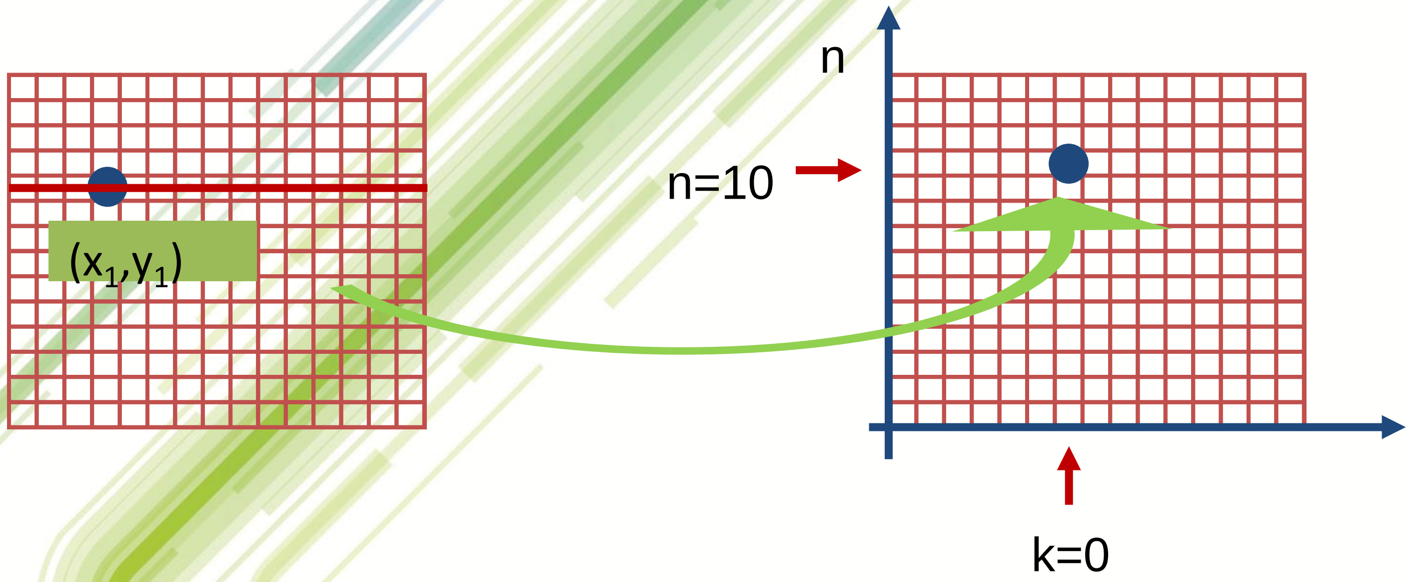
$n = -x_1 * k + y_1$



- Given a single point (x_1, y_1) in the input image, any line (k, n) through that point could be drawn.
  $y_1 = k.x_1 + n \rightarrow n = -x_1.k + y_1$
- All pairs (k, n) satisfying this linear dependence are possible

Hough transform for lines

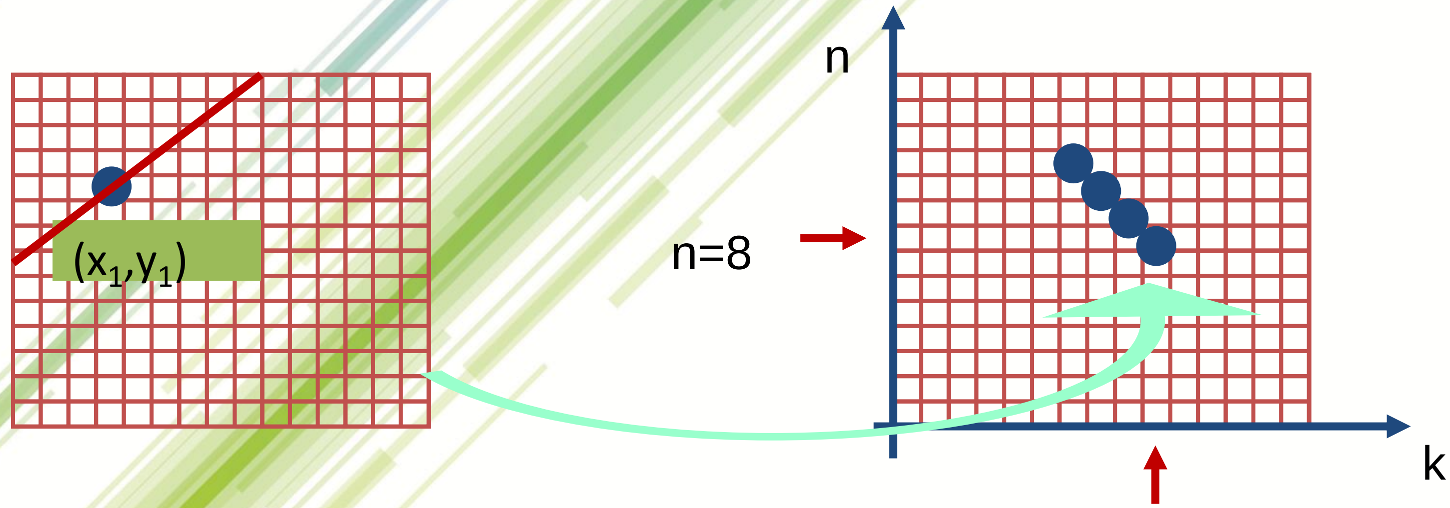
- Let $(x_1, y_1) = (4, 10)$.
- An arbitrary number of lines can be drawn through that point



- Imagine we draw a horizontal line, $k = 0$, $n = 10$, that is $y = 10$.
- The cell $(k,n) = (0, 10)$ gets (one) vote.

Hough transform for lines

– Let $(x_1, y_1) = (4, 10)$, the same point as before

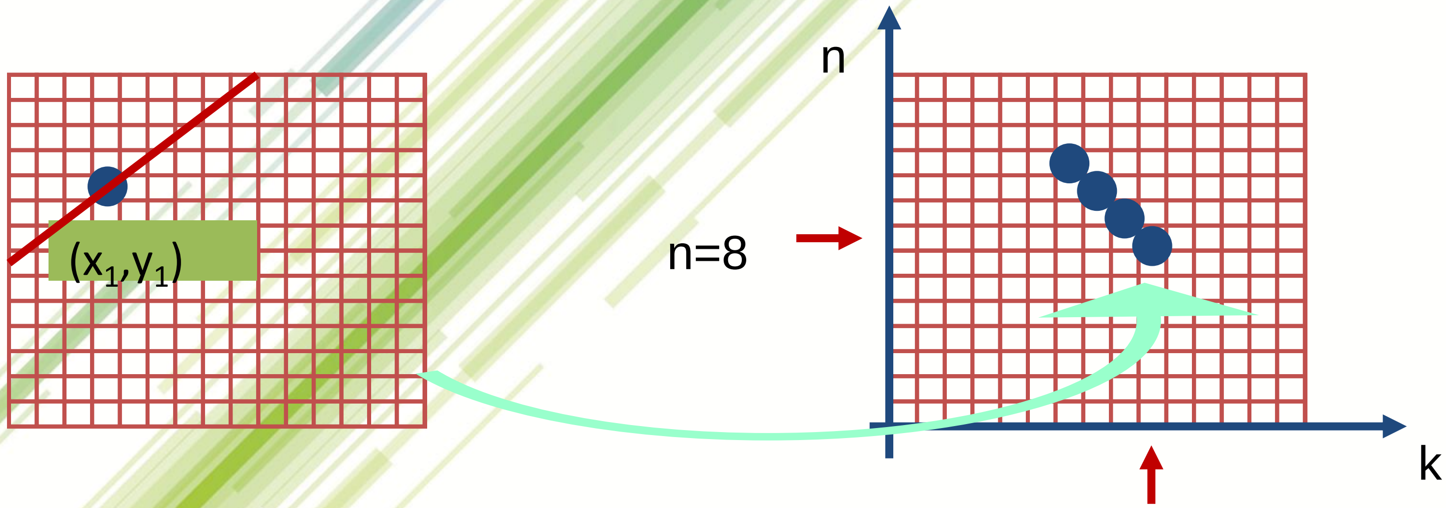


– Imagine we draw 4-th line, $k = 0.75$, $n = 7$, that is $y = 0.75\,x + 7$.

– Now the cell $(k,n) = (0.75,7)$ gets (one) vote.

Hough transform for lines

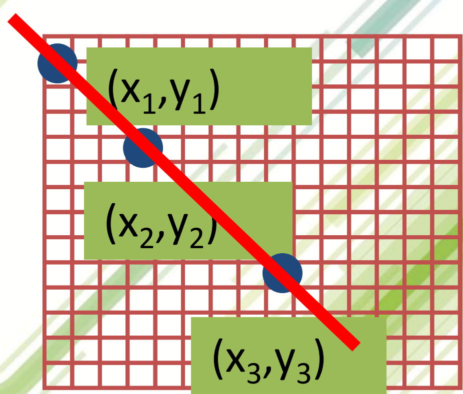
- Let $(x_1, y_1) = (4, 10)$, the same point as before

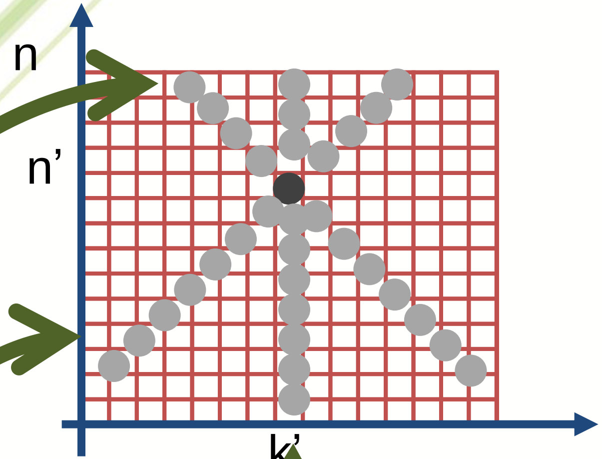


- Imagine we draw 4-th line, $k = 0.75$, $n = 7$, that is $y = 0.75\,x + 7$.
- Now the cell $(k,n) = (0.75,7)$ gets (one) vote.

Illustration: Hough transform for lines

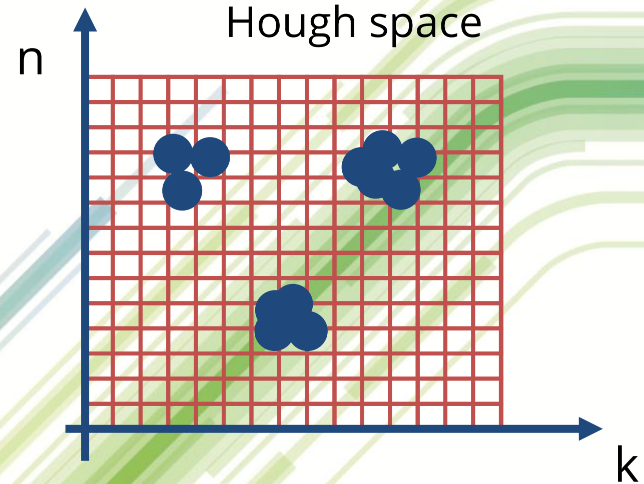
- Let 's have three points (x_1, y_1),(x_2, y_2) and (x_3, y_3)
- These points lie on a line. What happens in the Hough space?





- For each point in image, we can draw multiple lines in the HS, that satisfy the equations
  $n = -x_1 * k + y_1$ $n = -x_2 * k + y_2$ $n = -x_3 * k + y_3$
- Only one point in HS got 3 votes - the point (k', n')
- *Therefore, 3 points in image support the model y=k'*x+n'!*

Hough transform for lines



- In practical situations there are many straight line structures in the image. Each such line structure gives large number of votes to a particular accumulator cell in HS.
- But, due to noise in the image, there will be “clusters” of accumulators rather than a single accumulator cell having higher values than the rest of accumulators.
- Nevertheless, we can threshold HS to detect those clusters.
- Then, we can compute the centers of clusters to estimate the parameters of the model, k and n in this case.

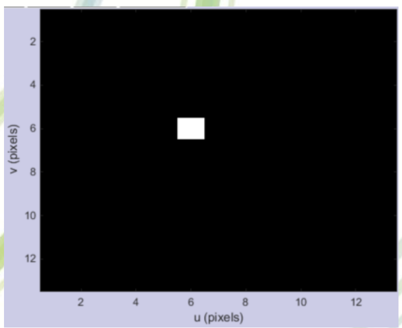
Hough transform for other structures

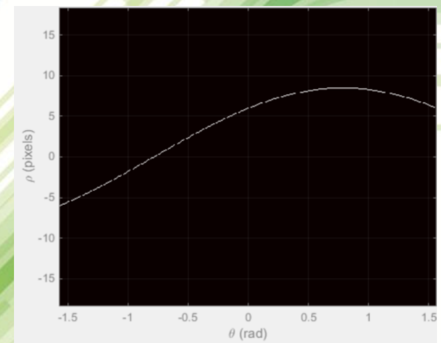
- Hough transform is based on a voting principle
- It can be used (generalized) for any parametric model, not just lines
  - Lines: f(x,y) = f(x,y, k, n) = f(x, y, q) = k.x + n – y = 0;
  - Circles: f(x,y) = f(x,y,x_0, y_0, r) = f(x,y,q) = (x-x_0)^2 $^+$ (y-y_0)^2 – r = 0;
- Algorithm:
  - Set H (acumulators) to zero
  - For each point (x, y) in the image
  - increment accumulators H if f(x,y,q) = 0;
  - H(q) = H(q) + 1; //or for some ‚delta' instead
  - Find local maxima of H
- Note:

for lines we never use model y=k*x+n! We prefer "normal" eq., ro = x . cos(theta) + y . sin(theta).

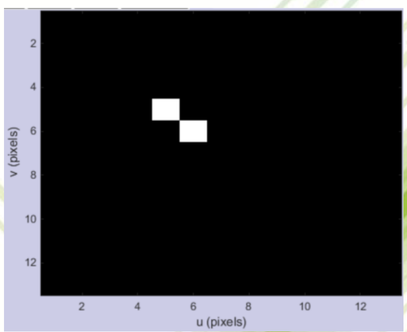
Hough transform for lines, normal eq.

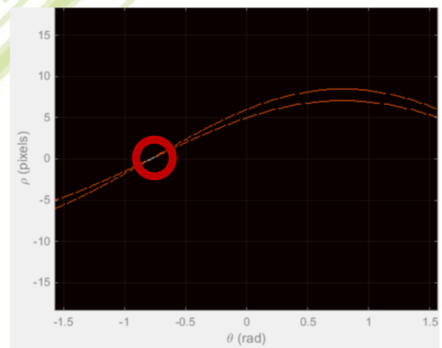
ro = x . cos(theta) + y . sin(theta)



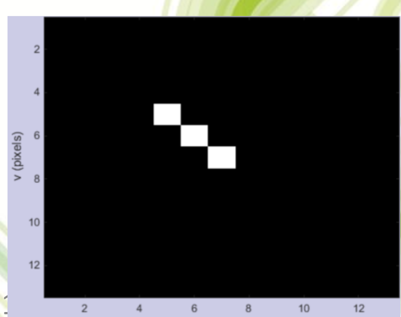


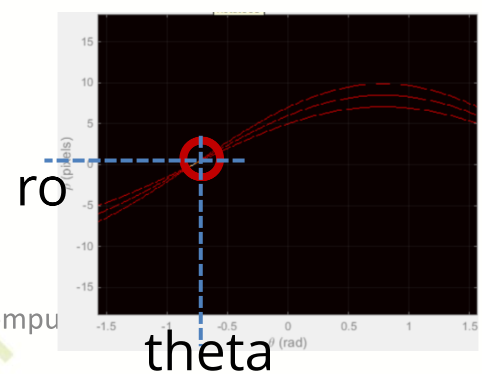
One pixel votes
(sinusoid)





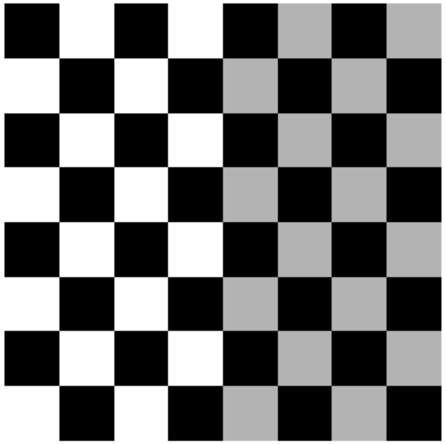
Two pixels' votes
(two sinusoids, two votes at intersection for a line)



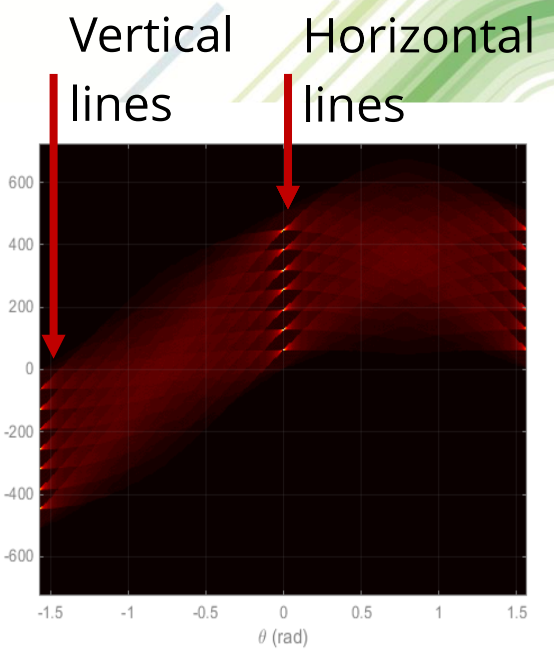


Three pixels votes
(all three pixels vote for Ro,theta of a line that they define)

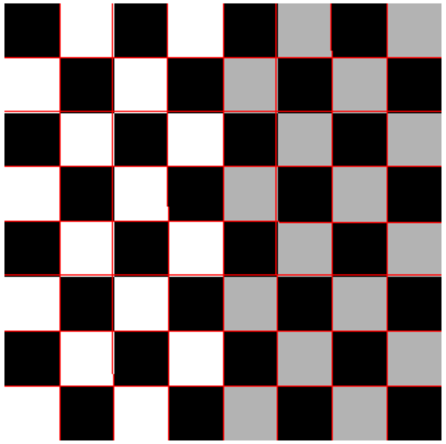
Example: Hough transform



Synthesized image

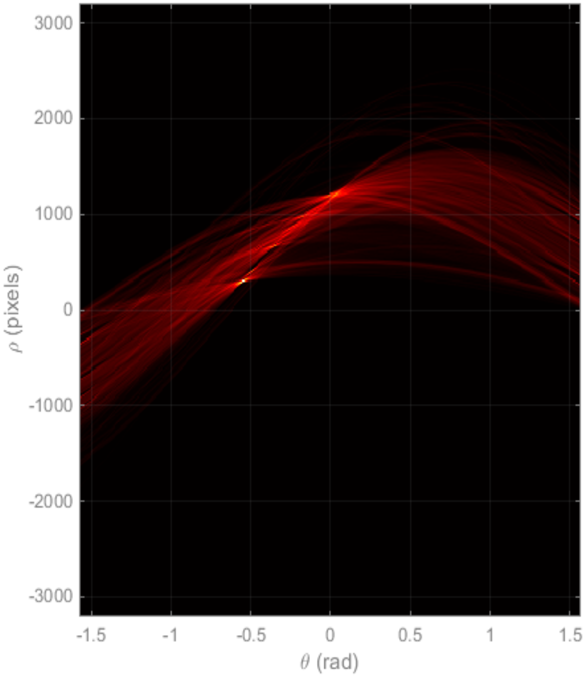


Hough on edge image produced by canny



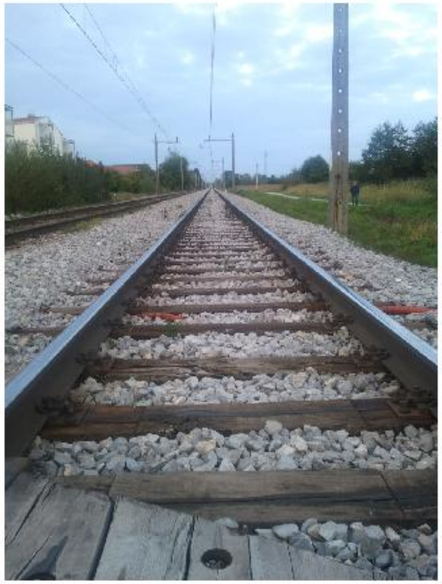
input image with lines overlaid

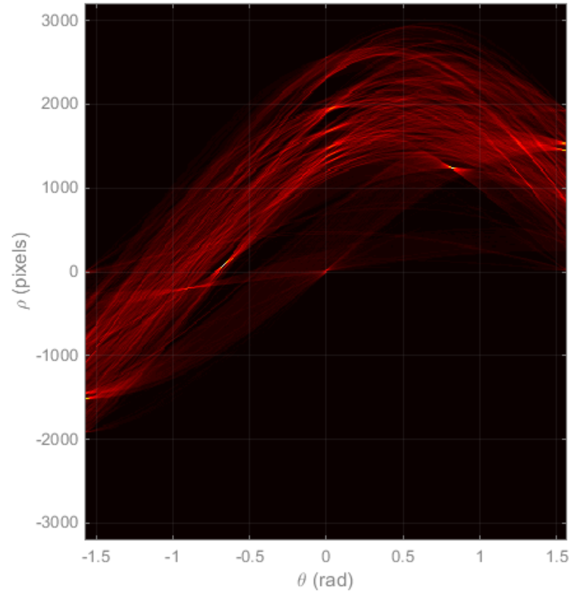
Example: Hough transform

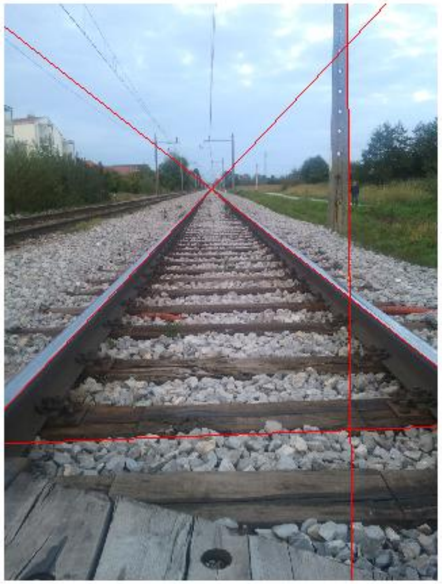


Original image -> rgb2gray -> canny -> Hough -> lines overlaid

Example: Hough transform



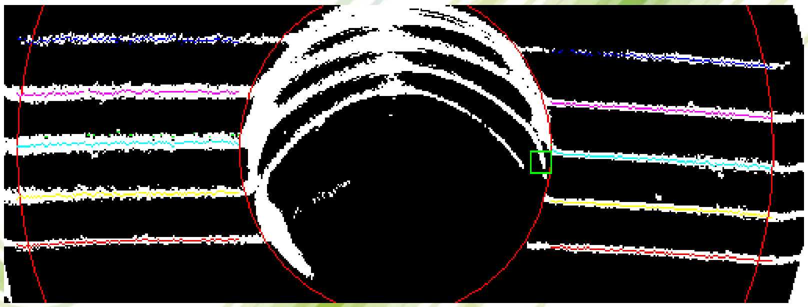


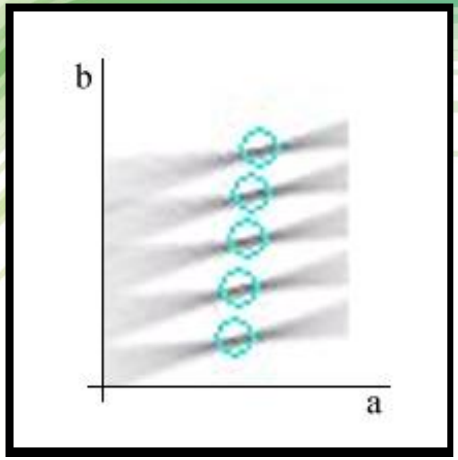


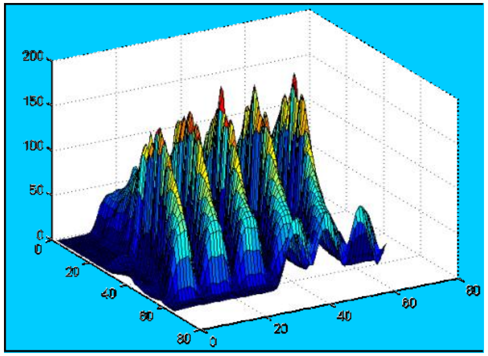
Original image -> rgb2gray -> canny -> Hough -> lines overlaid

Example: Hough transform

- Industrial application!







Multiresolution approach

- Noise filtering, edge detection, corner detection, …
  - all rely on pre-smoothing with Gaussian filter.

- Nevertheless, we have faced several times the same question:
  - „What sigma to use, and moreover,
  - how (based on what) to select an appropriate value of the sigma? parameter“
- Answer to this is multiresolution approach
  - Doing the processing for multiple sigmas, resolutions, etc.

Multiresolution vs. multiscale

- Multiresolution should be easy to grasp
  - Instead of processing image at the given resolution, we downsample it, e.g. to ½, ¼, and 1/8 of the original resolution
  - The image structures scale with the resolution
    - e.g. a circle having the radius of 50 pixels in the original image will have radius of 25, 12.5 and 6.25 pixels, respectively
  - Consequence: effectively the "local neighborhood" becomes larger.
    - So a 3x3 smoothing kernel effectively covers area of 24x24 pixels, if we reduce resolution to 1/8!
  - Benefit: we don't need to design multiple size kernels for the same task, we change image resolution instead.

Multiresolution vs. multiscale

- Multiscale is a bit different, but generally used to achieve similar effect
  - Remember – when *resampling/resizing image* we have to *filter it properly, to remove higher frequencies.*
  - What if we do filtering, but do not follow up with resampling? Then our image keeps nominally the same resolution
  - But the high frequency contents is gone
- Sometimes it is preferable to keep image resolution intact and just smooth/filter it appropriately
  - This way we changed the scale but not the resolution
  - There is no influence on the size of local neighborhood

More on multiresolution/multiscale

- In theory, appropriate sigma should be easy to find:
  - the one that filters out “all high frequency (HF) noise” while preserving the signal.
  - Unfortunately, this is in general impossible to do, because signal and noise components are (additively) intermixed.
  - Thus, it also filters out HF components of signal, and therefore it also blurs the image.
- “optimal” sigma is the one that
  - according to the predefined criteria filters out HF noise as much as possible, while preserving HF content of interest.
  - There is an obvious trade-off between preserving the signal and suppressing the noise.

More on multiresolution/multiscale

- And, although filtering with larger sigma improves edge detection, it also worsens the localization.
  - Obviously, there is a trade-off between the quality of detection and the precision of localization.
- All in all, the selection of appropriate sigma is never trivial.
  - Therefore, we strive for, in same sense, optimal solution.
  - But there is more…

Multiresolution approach

- Namely, sigma effectively controls the *effective resolution*, or *scale* of an image.
  - As sigma increases the effective image resolution decreases.
  - Q: what is a proper scale to represent an image anyway?
  - A: Simple question, but there is no simple answer.

  - Because there exists a suitable, but only *limited range of scales* on which some *phenomenon appears*, and consequently, on which that *phenomenon can be observed.*

Multiresolution approach

- – Structures that can be observed on one resolution (scale) cannot be observed on some other resolution.
- – Moreover, structures that are present at one resolution (scale) are not even present at some other resolution.

- Think of:
  - – shape of clouds, …
  - – forest, trees, branches, leaves, …
  - – printed text, size (scale) for reading, scale for inspecting the quality of print, …
  - – fabric, fabric textures, defects in fabric, …
- Once again
  - – there is always an appropriate, but limited range on scales on which some phenomenon appears, and can be observed.

Multiresolution approach

- There is no single solution to the selection of scale (controlled by sigma).
- The selection of scale largely depends on the purpose!
  - If we want to observe trees, then there is an appropriate scale of observation.
  - If we wanted to observe leaves, then there would be another appropriate scale of observation.
- We humans are quite good on selecting an appropriate scale (for observing phenomena we are familiar with, i.e. have previous knowledge).
- And then, we tend to develop solutions/technology for that.
- In situations where there is no prior knowledge of scale a natural choice is to represent the image at *all possible scales.*

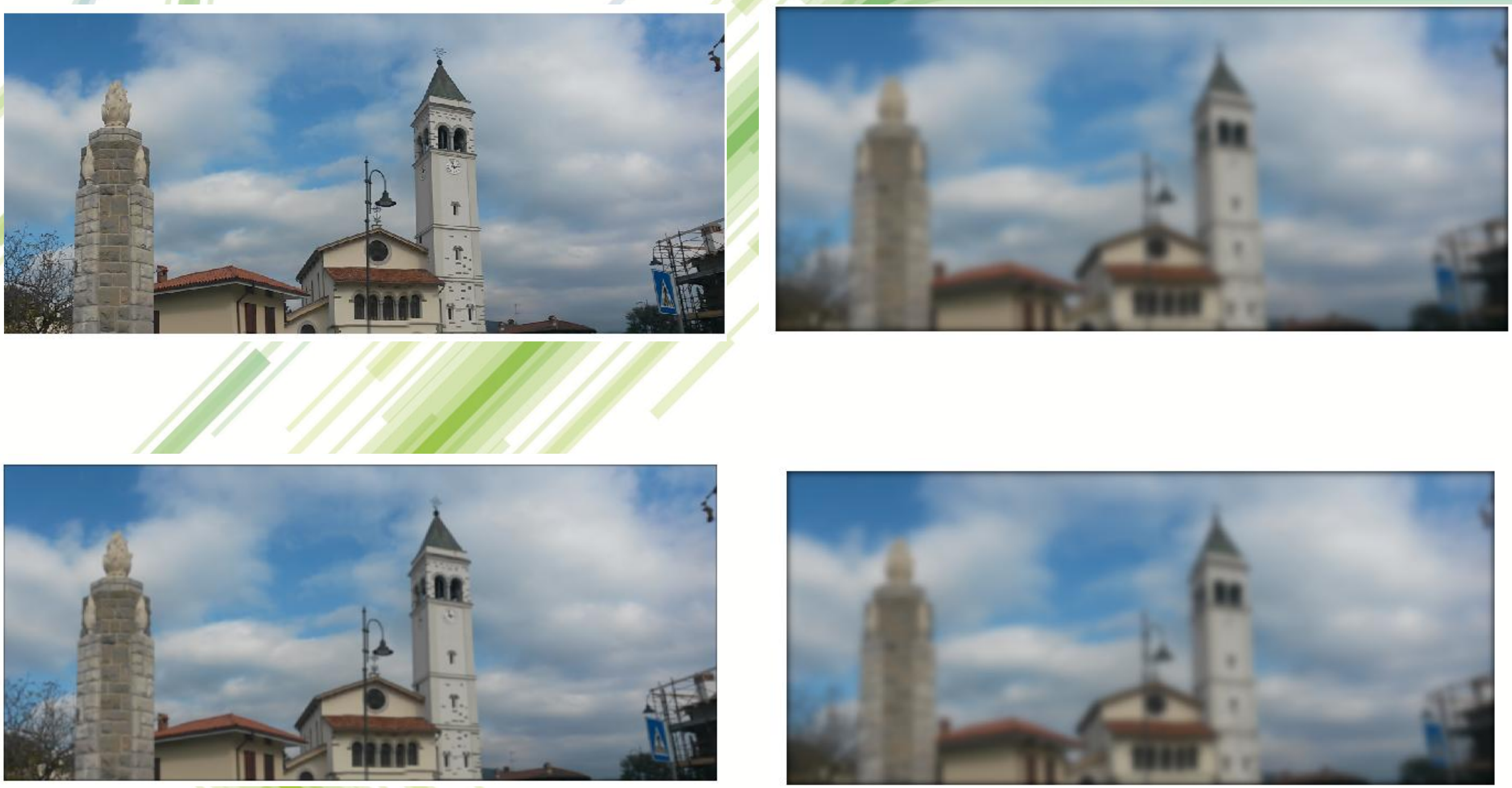
Multiresolution approach

- Multiresolution approaches have many uses in CV and IP:
  - Image coding
  - Image matching
  - Image registration
  - Image segmentation
- Initiators & developers of the theory:
  - P. Burt, E. Adelson,
  - A. Witkin,
  - T. Lindeberg,
  - J. Koenderink,
  - P. Perona, J. Malik,
  - J. Weickert, ...

Multiresolution approach

- As an example:
  - Original image is filtered with Gaussian of increasing sigma
  - The result is a ‚stack' of images of lower and lower resolution (scale)
  - Detect edges and corners at ‚all' resolution (this could be combined with filtering)
  - Combine the results (The question is how)

Multiresolution approach



Multiresolution approach

Another view of multiresolution approach

Image pyramid!

Questions?