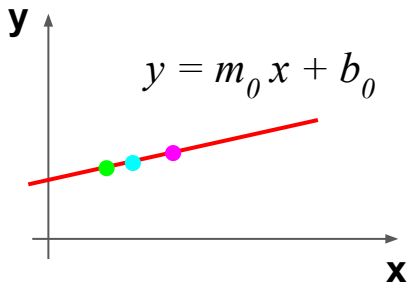
Lab 04

Basic Image Processing
Fall 2019

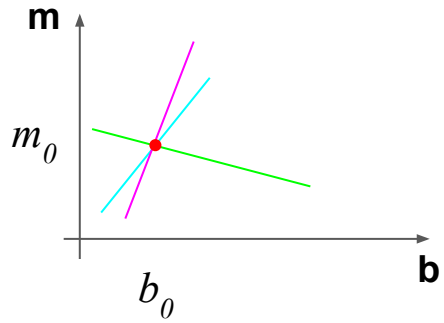
Part 1
Line detection with Hough Transform

Hough Transform – Introducing the Hough space

image space

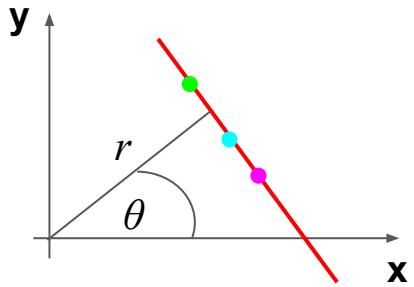


m-b space

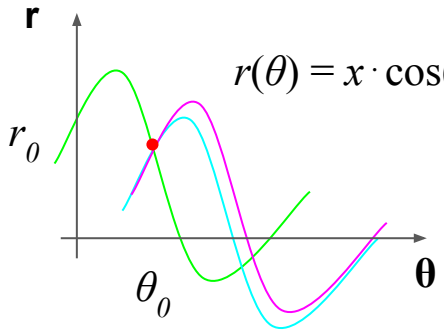


everything OK,
except when m=∞

image space

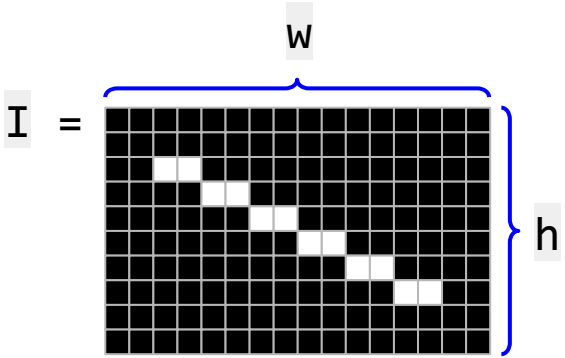


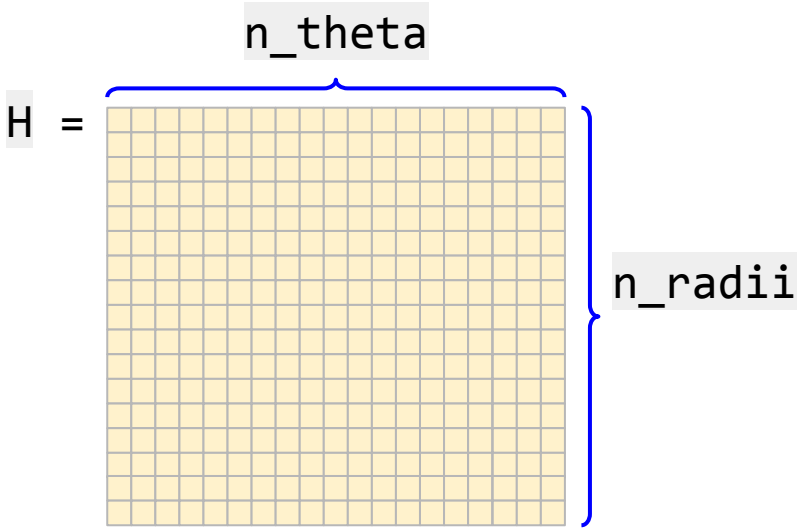
Hough space (r-θ space)



everything OK,
no exception

Hough Transform – The discretized Hough space





The origin of the image is at the top left corner. The maximal length line segment on this image is the diagonal, therefore

$$r_{\max} = \sqrt{h^2 + w^2}$$

We want the Hough space to be a matrix. For this we have to discretize the angle and radius values. This is done with a resolution of 1, meaning that the number of columns is `n_theta` = 180 as θ goes from $1°$ to $180°$, and `n_radii` has the same value as $r_{\max}$ and hence the resolution of the matrix along this dimension is 1 pixel.

Hough Transform – Algorithm

Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

For each `theta` value `theta = 1:180`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

Do the quantization (rounding) of the radius value

Store the vote (increment `H(r, theta)` by one)

Return

Return the matrix containing the votes.

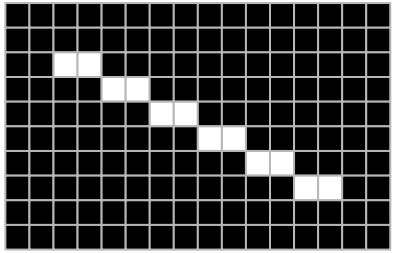
Example: running of the algorithm

Initialization

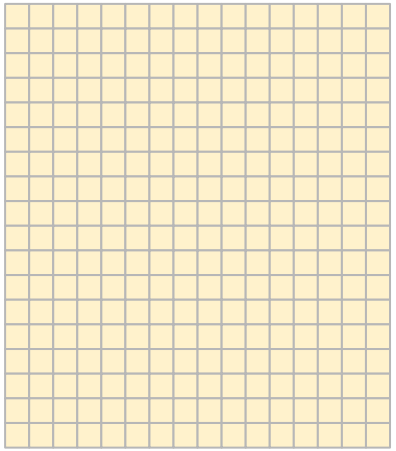
For all `r` and `theta` initialize `H(r, theta) = 0`

The size of the input image was used to determine the maximal possible radius, which gives the number of rows in H.

`I` =

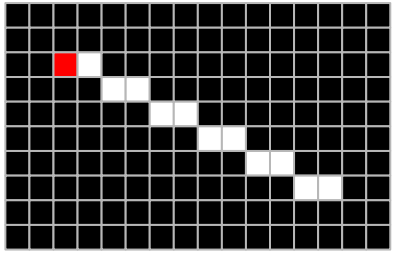


`H` =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

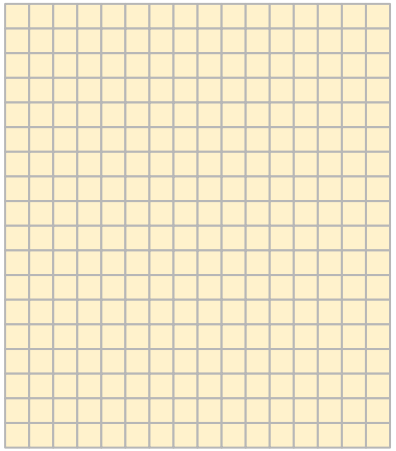
Voting

For each edge point `I(x, y)` in the image

In this first iteration the first edge point is selected; it is

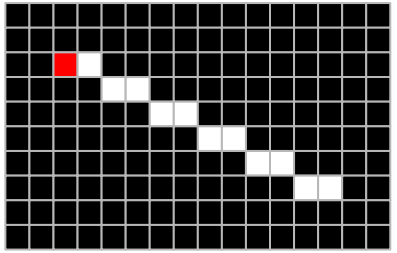
`I(3, 3)`

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

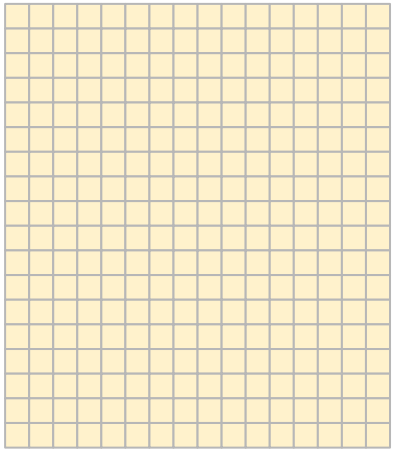
`I(3, 3)`

For each `theta` value `theta = 1:180`

In this first iteration the first θ value is selected; it is

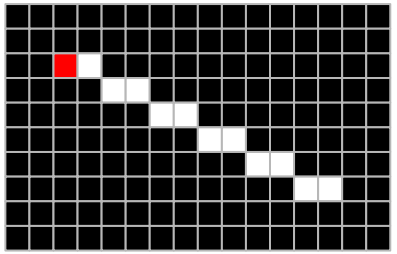
`theta = 1`

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

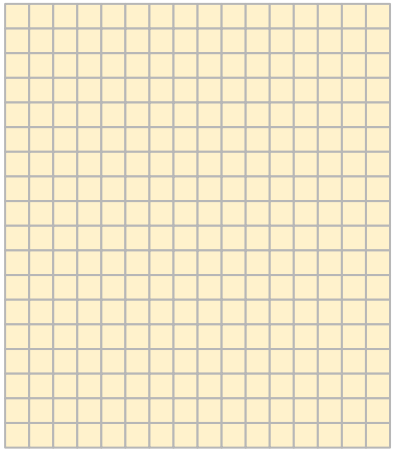
`theta = 1`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

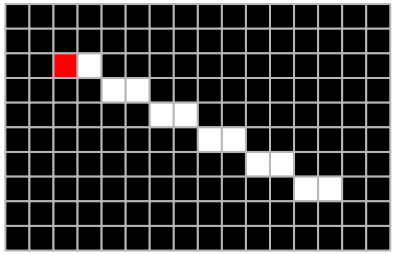
$$r(1°) = 3 \cdot \cos(1°) + 3 \cdot \sin(1°) = 3.05$$

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

`theta = 1`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

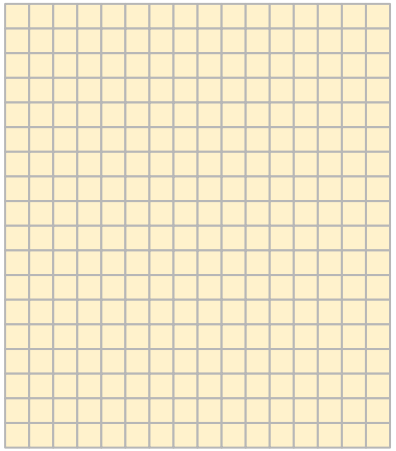
$r(1°) = 3.05$

Do the quantization (rounding) of the radius value

The computed radius value is rounded

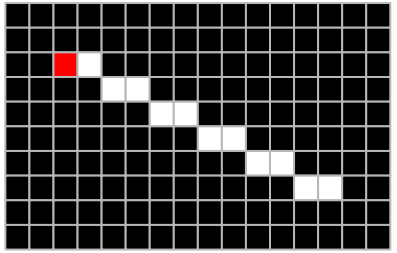
`r = 3`

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

`theta = 1`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

$r(1°) = 3.05$

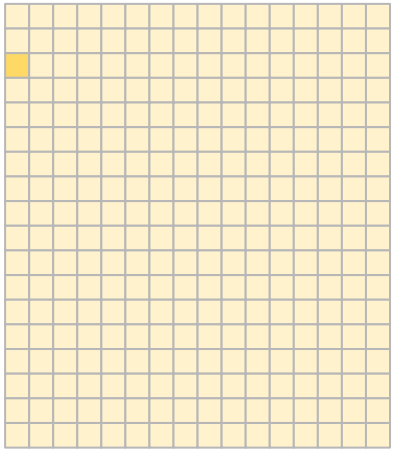
Do the quantization (rounding) of the radius value

`r = 3`

Store the vote (increment `H(r, theta)` by one)

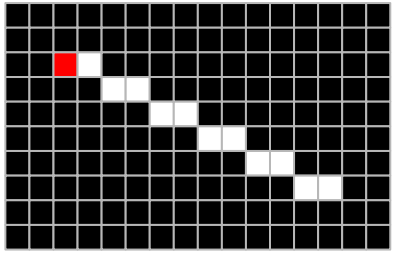
The element at `H(3, 1)` is selected and its value is incremented.

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

`theta = 1`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

$r(1°) = 3.05$

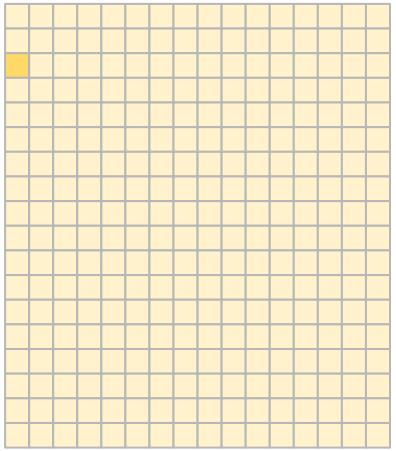
Do the quantization (rounding) of the radius value

`r = 3`

Store the vote (increment `H(r, theta)` by one)

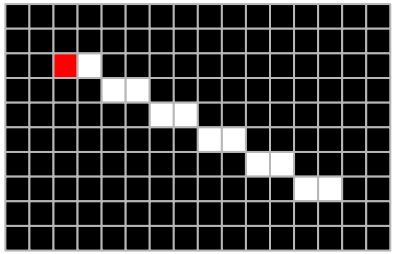
Innermost loop core is done, do the next iteration!

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

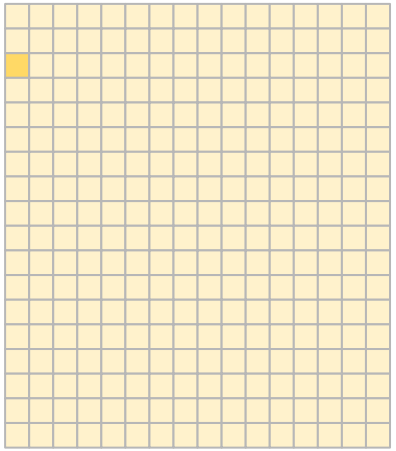
`I(3, 3)`

For each `theta` value `theta = 1:180`

In this second iteration the next θ value is selected; it is

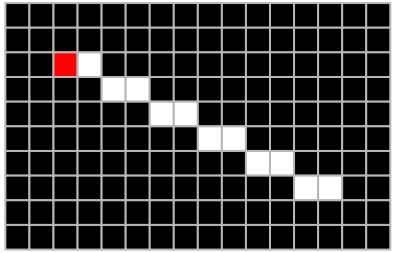
`theta = 2`

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

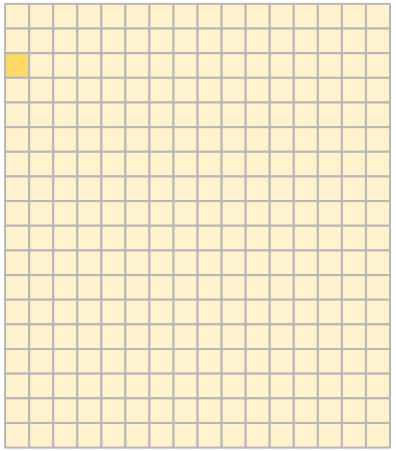
`theta = 2`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

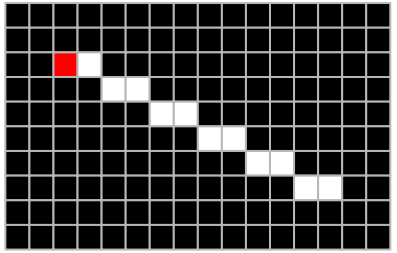
$$r(2°) = 3 \cdot \cos(2°) + 3 \cdot \sin(2°) = 3.10$$

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

`theta = 2`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

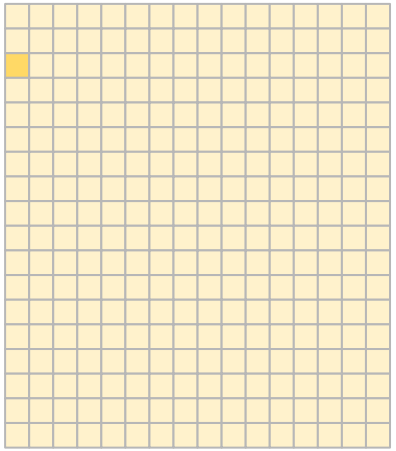
$r(1°) = 3.10$

Do the quantization (rounding) of the radius value

The computed radius value is rounded

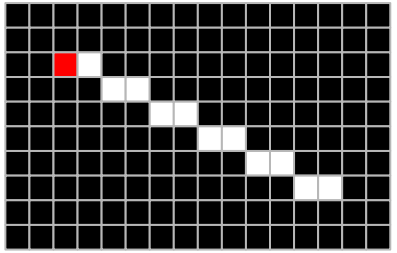
`r = 3`

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

`theta = 2`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

$r(1°) = 3.10$

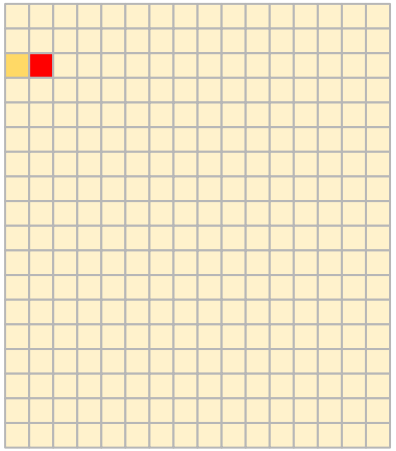
Do the quantization (rounding) of the radius value

`r = 3`

Store the vote (increment `H(r, theta)` by one)

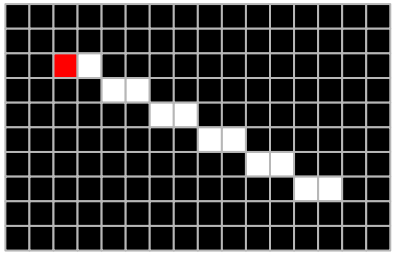
The element at `H(3, 2)` is selected and its value is incremented.

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

`theta = 2`

Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

$r(1°) = 3.10$

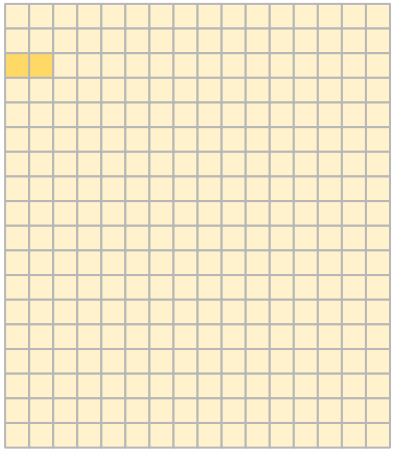
Do the quantization (rounding) of the radius value

`r = 3`

Store the vote (increment `H(r, theta)` by one)

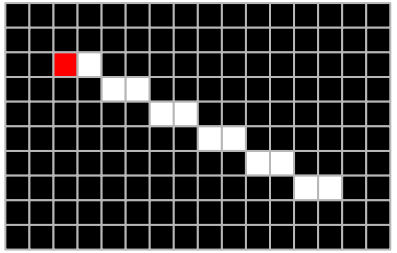
Innermost loop core is done, do the next iteration!

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 3)`

For each `theta` value `theta = 1:180`

Calculate the radius using the formula

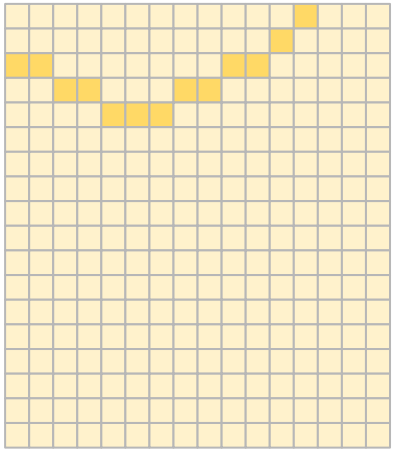
$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

Do the quantization (rounding) of the radius value

Store the vote (increment `H(r, theta)` by one)

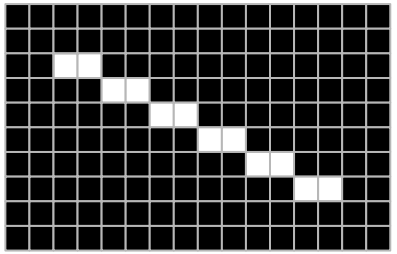
After the completion of all iterations with the `theta` angle the Hough matrix is filled with votes coming from the edge pixel at `I(3, 3)`

H =



Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

For each `theta` value `theta = 1:180`

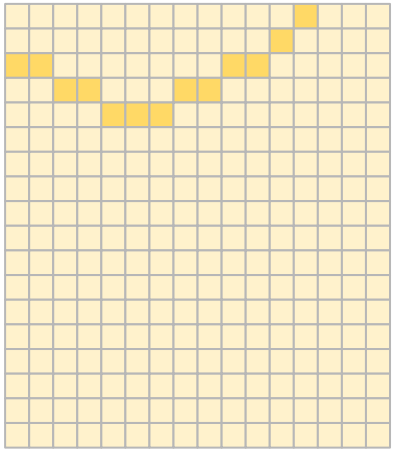
Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

Do the quantization (rounding) of the radius value

Store the vote (increment `H(r, theta)` by one)

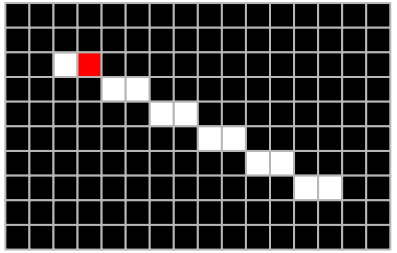
H =



The next iteration of the outer loop continues this process with the next edge pixel: `I(3, 4)`

Example: running of the algorithm

I =



Initialization

For all `r` and `theta` initialize `H(r, theta) = 0`

Voting

For each edge point `I(x, y)` in the image

`I(3, 4)`

For each `theta` value `theta = 1:180`

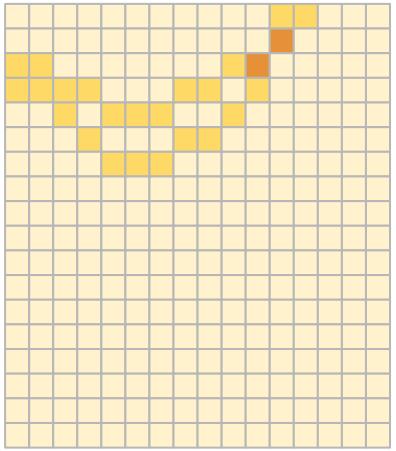
Calculate the radius using the formula

$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

Do the quantization (rounding) of the radius value

Store the vote (increment `H(r, theta)` by one)

H =



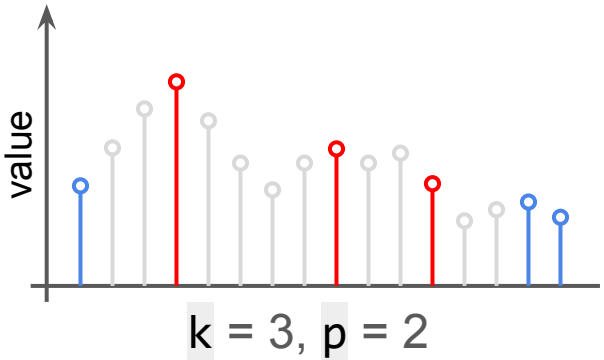
After the completion of all iterations with the `theta` angle the Hough matrix is filled with votes coming from the edge pixel at `I(3, 4)`

These votes are combined with the votes coming from the previous edge pixel, so now there are values in `H` that were incremented twice.

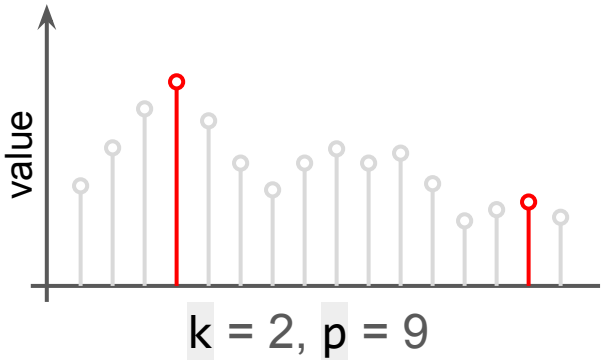
Non-maximum suppression – Goal

The use of non-maximum suppression is to process data containing multiple local maxima points and return the ‘true’ maxima values (and their locations).

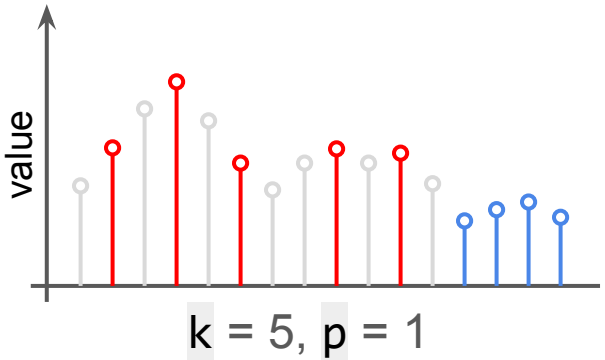
The problem: Given (a usually noisy) data we want to find the first k maxima points where the distance between any two maxima points is greater than p.



k = 3, p = 2



k = 2, p = 9



k = 5, p = 1

Non-maximum suppression – Algorithm

Initialization

Initialize the array of the found maxima points (it has `k` elements).

Iterative counting

While `k` is not zero

- Find the global maximum in the data
- Put this maximum point into the return array
- Suppress the maximum point and all the points in its radius `p` neighborhood
- Decrease `k` by 1

Return

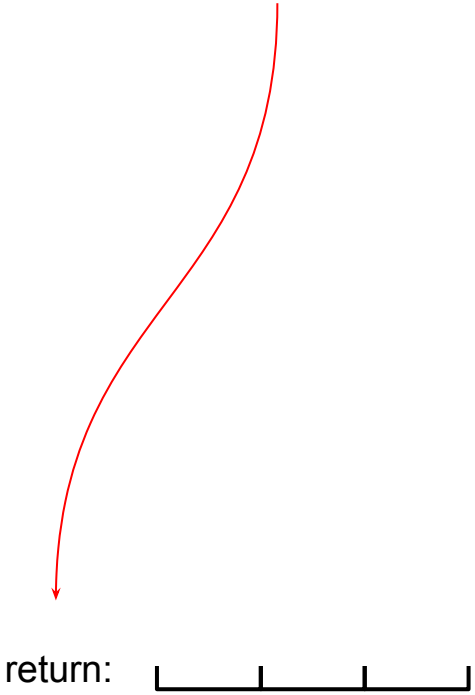
Return the array containing the maxima points.

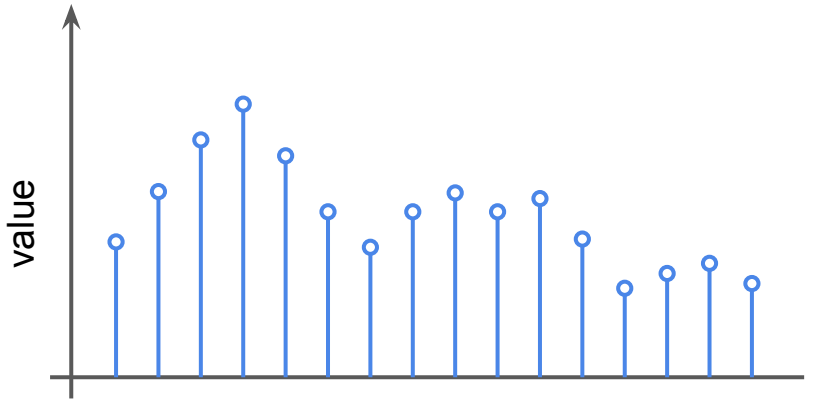
Example: running of the algorithm

Initialization

Initialize the array of the found maxima points (it has k elements).

$k = 3, p = 2$





Example: running of the algorithm

Initialization

Initialize the array of the found maxima points (it has `k` elements).

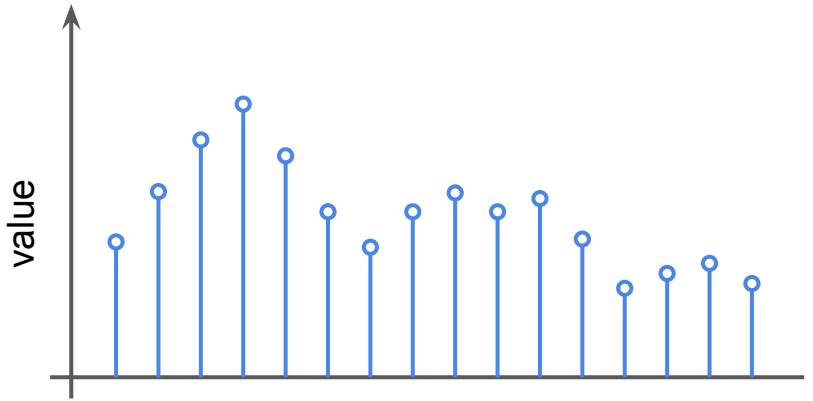
$k = 3,\ p = 2$

Iterative counting

While `k` is not zero

```
k = 3
```

return:



Example: running of the algorithm

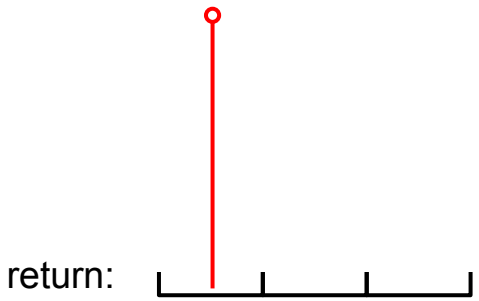
Initialization
Initialize the array of the found maxima points (it has `k` elements).

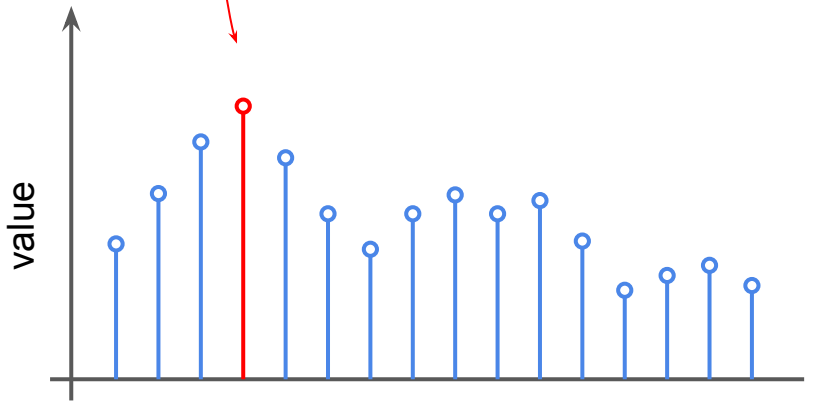
$k = 3, p = 2$

Iterative counting
While `k` is not zero

```
k = 3
```

Find the global maximum in the data
Put this maximum point into the return array





Example: running of the algorithm

Initialization

Initialize the array of the found maxima points (it has `k` elements).

$k = 3,\ p = 2$

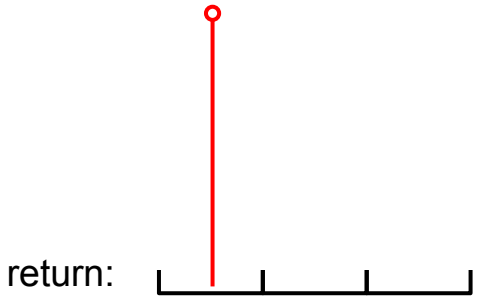
Iterative counting

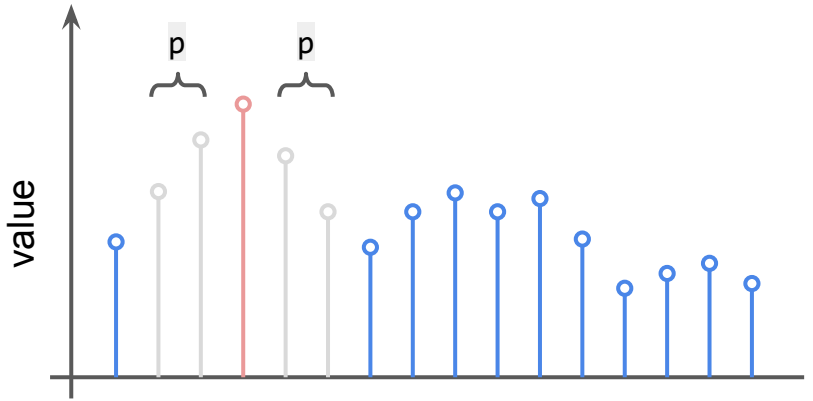
```
k = 3
```

While `k` is not zero

- Find the global maximum in the data
- Put this maximum point into the return array
- Suppress the maximum point and all the points in its neighborhood with a radius `p`
- Decrease `k` by 1.

```
k := 2
```





Example: running of the algorithm

Initialization

Initialize the array of the found maxima points (it has `k` elements).

$k = 3,\ p = 2$

Iterative counting

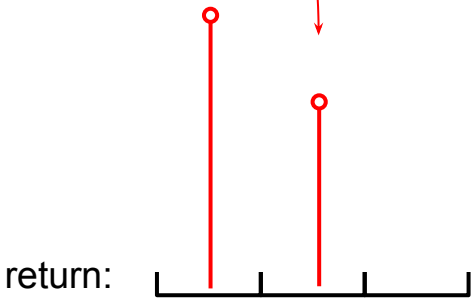
`k = 2`

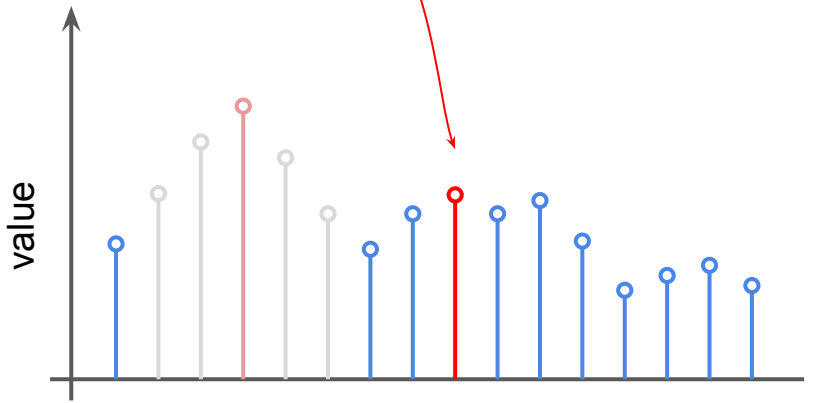
While `k` is not zero

Find the global maximum in the data

Put this maximum point into the return array

return:





Example: running of the algorithm

Initialization

Initialize the array of the found maxima points (it has `k` elements).

$k = 3,\ p = 2$

Iterative counting

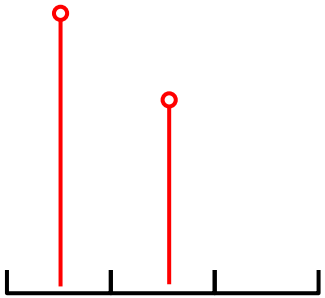
```
k = 2
```

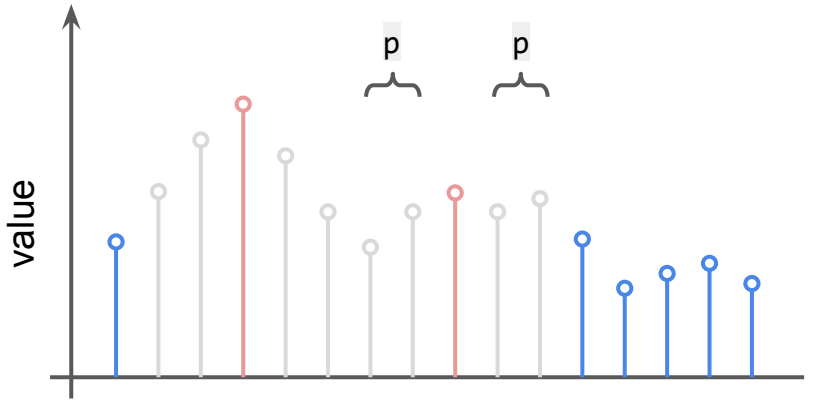
While `k` is not zero

- Find the global maximum in the data
- Put this maximum point into the return array
- Suppress the maximum point and all the points in its neighborhood with a radius `p`
- Decrease `k` by 1.

```
k := 1
```

return:





Example: running of the algorithm

Initialization

Initialize the array of the found maxima points (it has `k` elements).

$k = 3,\ p = 2$

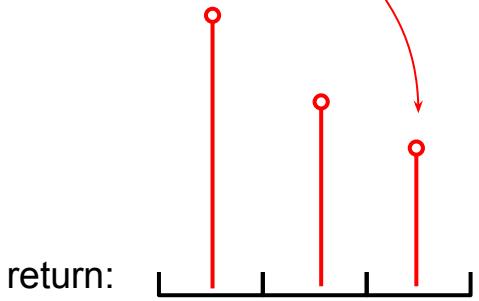
Iterative counting

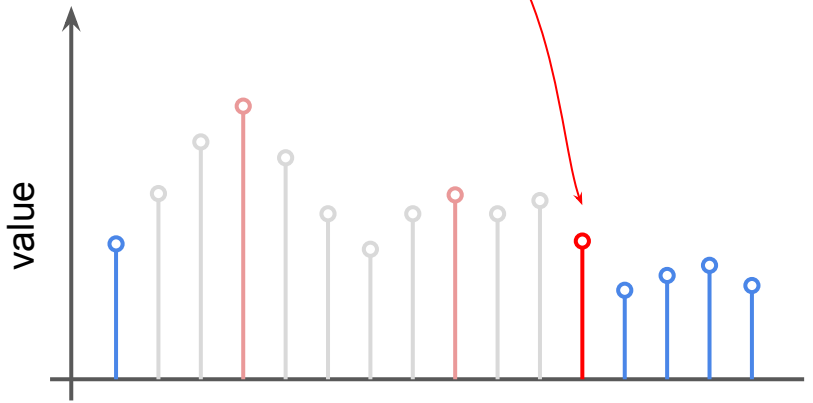
```
k = 1
```

While `k` is not zero

Find the global maximum in the data

Put this maximum point into the return array





Example: running of the algorithm

Initialization

Initialize the array of the found maxima points (it has `k` elements).

$k = 3,\ p = 2$

Iterative counting

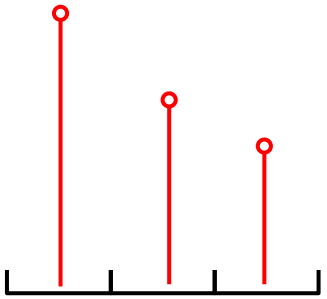
```
k = 1
```

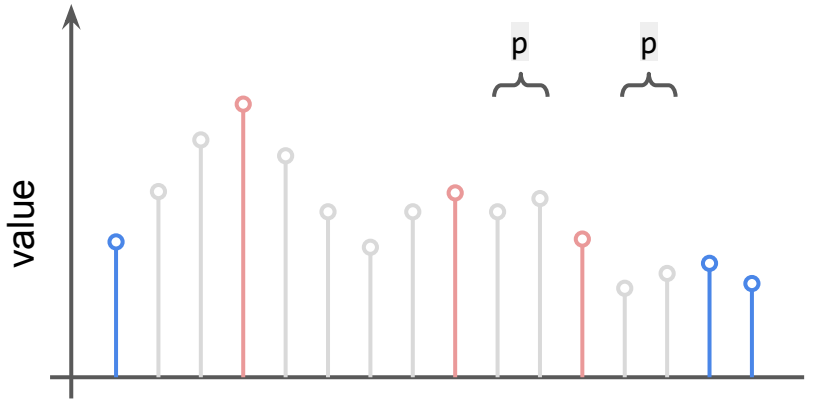
While `k` is not zero

- Find the global maximum in the data
- Put this maximum point into the return array
- Suppress the maximum point and all the points in its neighborhood with a radius `p`
- Decrease `k` by 1.

```
k := 0
```

return:





Example: running of the algorithm

Initialization

Initialize the array of the found maxima points (it has k elements).

k = 3, p = 2

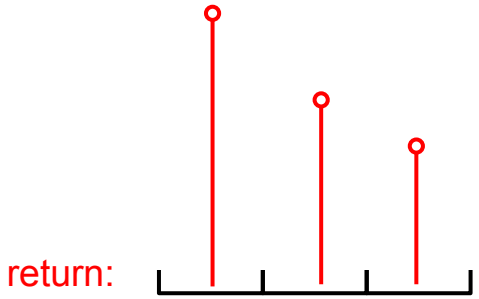
Iterative counting

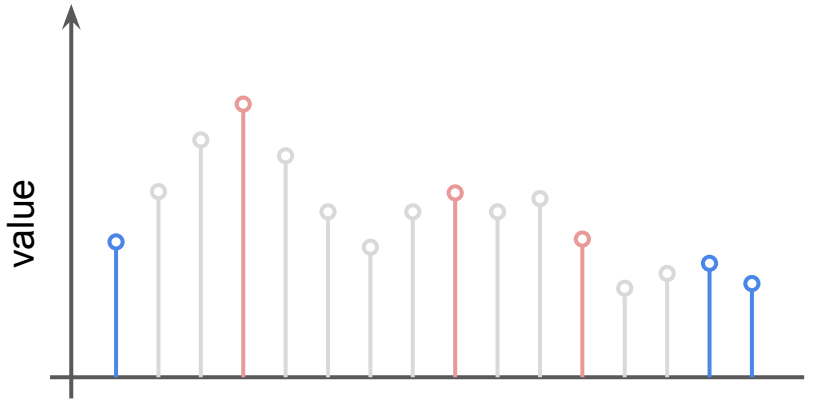
While k is not zero

- Find the global maximum in the data
- Put this maximum point into the return array
- Suppress the maximum point and all the points in its neighborhood with a radius p
- Decrease k by 1.

Return

Return the array containing the maxima points.





Now please

download the ‘Lab 04’ code package

from the

submission system

Exercise 1

Implement the function `my_hough` in which:

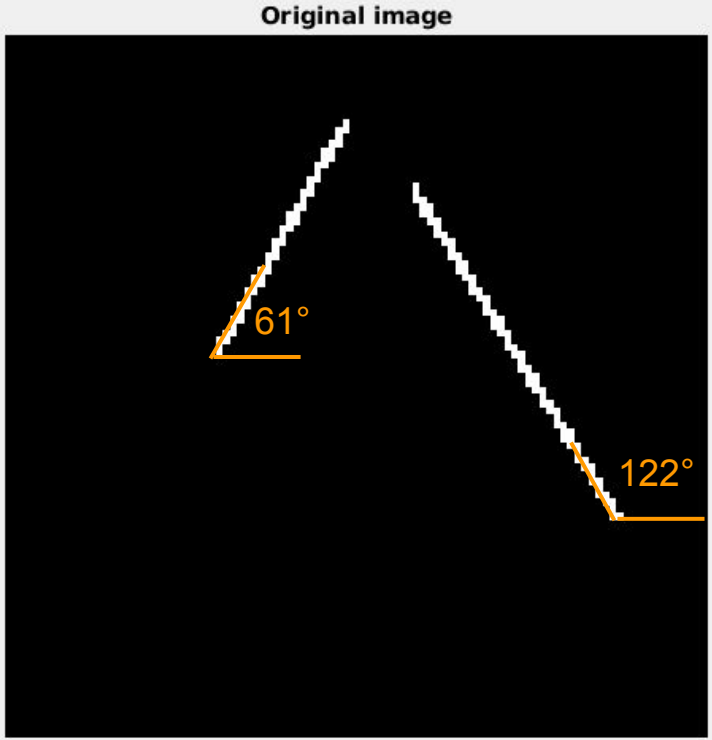
Realize the Hough Transform algorithm as described on Slide 5.

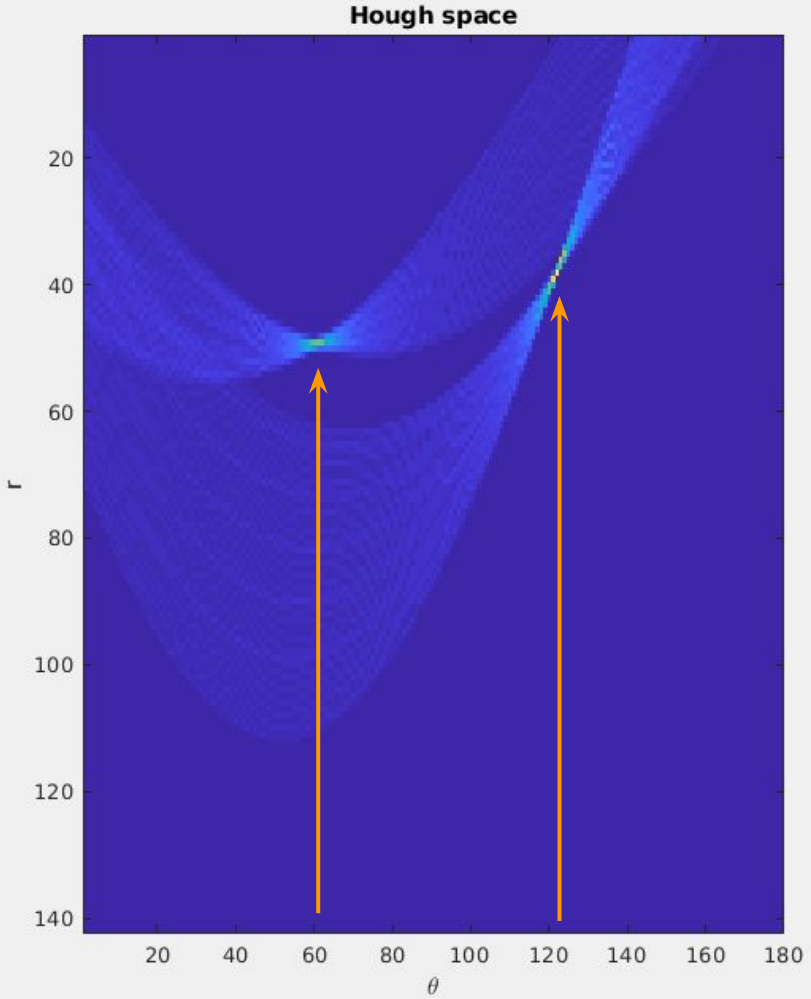
- Initialize the `H` matrix, where
  - the number of rows is the longest possible `r` radius on your original image (diagonal)
  - the number of columns is 180, referring the range of the angle `theta` $\in$ [1, 180]
- Iterate through your input image (`input_img`) with two (nested) `for` loops, and compute the (rounded) `r` radius at every nonzero pixel with all the possible θ values. Increment `H` at the appropriate location.
- After processing every edge pixel, return the Hough matrix.

Since the Hough transformation is applied on binary edge images, you can be sure that the input image is a black-and-white 0, 1 logical binary matrix.

Test your function with `script1.m`

The comments in amber are not part of the Matlab figure.





Exercise 2

Implement the function `non_max_sup` which has 3 input parameters:

- `H`: input matrix
- `k`: number of maxima points, whose neighboring regions should be suppressed,
- `p`: the radius of the region around a maximum to be suppressed.

The algorithm to be implemented: `while k > 0` do the followings

- find the maximum value of your Hough space array (`H`), collect its `r` and `theta` index in `r_vect` and `t_vect` arrays,
- replace the values in the [`-p`, `p`] neighborhood of the maximum point with zeros
- decrease `k`

See next slide for tips & tricks!

Exercise 2 – continued

Practical things to consider

there is a function called `ind2sub` which translates a linear indexing coordinate to a 2D one. You can use this trick for finding the location of the global maximum.

To avoid illegal indices when replacing the elements of `H`, use only integers >= 1

if `H(x_n, y_n)` is the center of the neighborhood

then `H(x_1:x_2, y_1:y_2) = 0`

where

`x_1` = *maximum of* `{ 1; x_n-p}`

`x_2` = *minimum of* `{size(H, 1); x_n+p}`

`y_1` = …

`y_2` = …

Test your function with `script2.m`

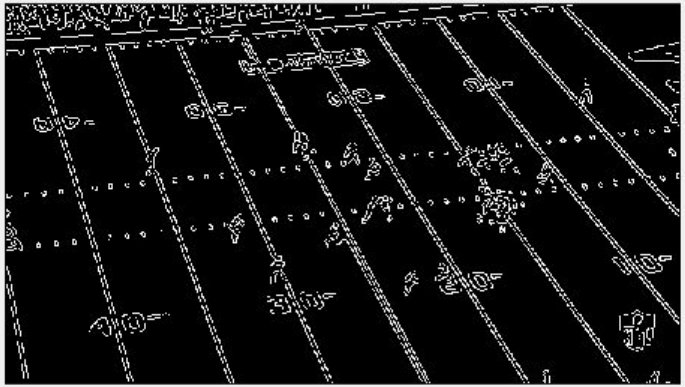
Exercise 3

After implementing the Hough algorithm and the non-maximum suppression, please open `script3.m` and try to understand what is happening there.

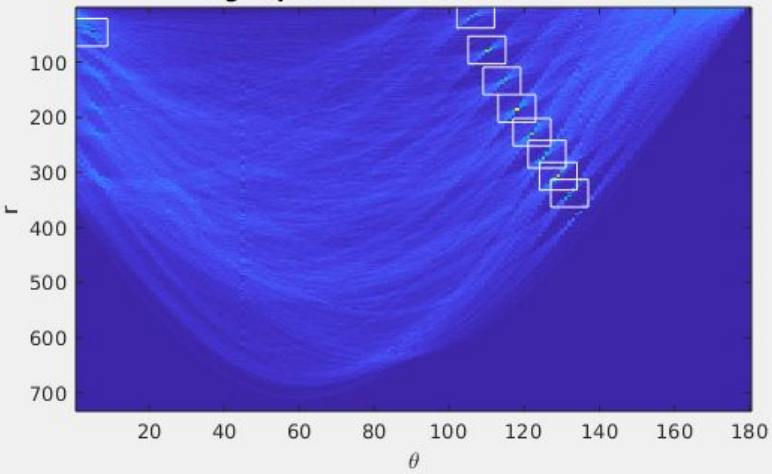
Run the script, examine the result and try to adjust the parameters to get something similar to the result presented on the next slide.

Original image

Canny edges result (th=0.17)



Hough space and the selected maxima



Detected lines

Part 2
Image enhancement

The Histogram of an Image

- **Histogram**:

  $h(k)$ = the number of pixels on the image with value k.

Original Image*

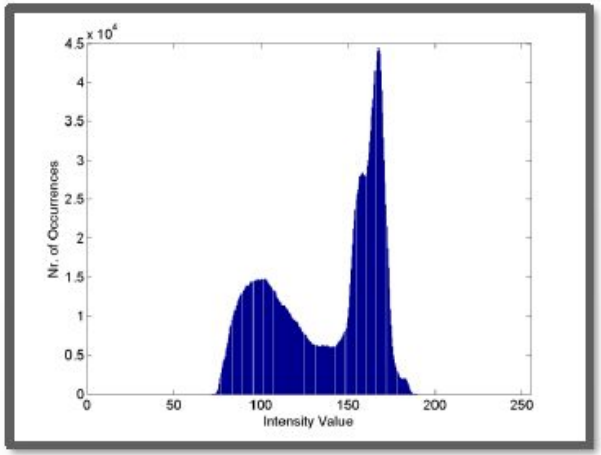


Image Histogram

- The histogram normalized with the total number of pixels gives us the ***probability density function*** of the intensity values.

* Modified version of Riverscape with Ferry by Salomon van Ruysdael (1639)

Histogram Transformations

- **Histogram Stretching**:
  - Based on the histogram we can see that the image does not use the whole range of possible intensities:
    - Minimum intensity level: 72
    - Maximum intensity level: 190
  - With the following transformation we can stretch the intensity values so they use the whole available range:

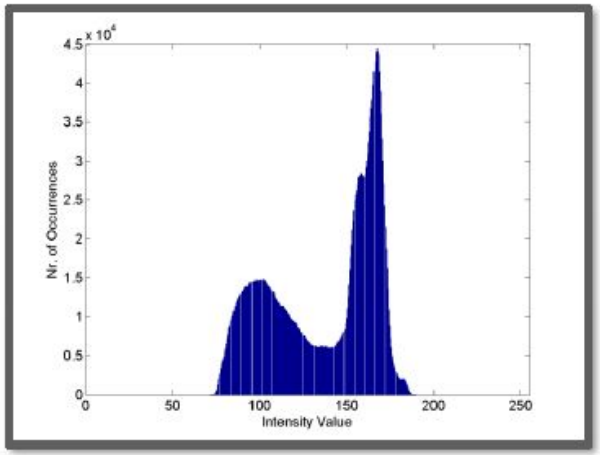


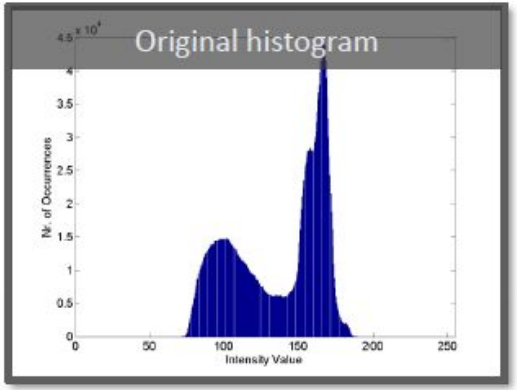
Image Histogram

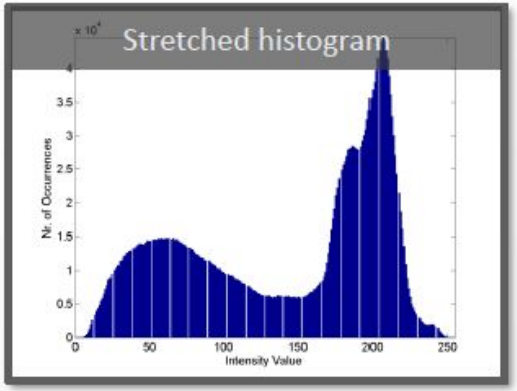
$$y(n_1, n_2) = \frac{255}{x_{\max} - x_{\min}} \cdot (x(n_1, n_2) - x_{\min})$$

$$x_{\max} = \max_{n_1, n_2}(x(n_1, n_2)) \qquad x_{\min} = \min_{n_1, n_2}(x(n_1, n_2))$$

Histogram Transformations

- **Histogram Stretching:**





Point-wise Intensity Transformation

- **Log transformation:** $y(n_1, n_2) = c \cdot \log\left(x(n_1, n_2) + 1\right)$
  - Expands low and compresses high pixel value range

Original Image*

Log Image

Log Image
after histogram stretching

* Abbaye du Thoronet by Lucien Hervé (1951)

Exercise 4

Implement the function `calc_hist_vector` in which:

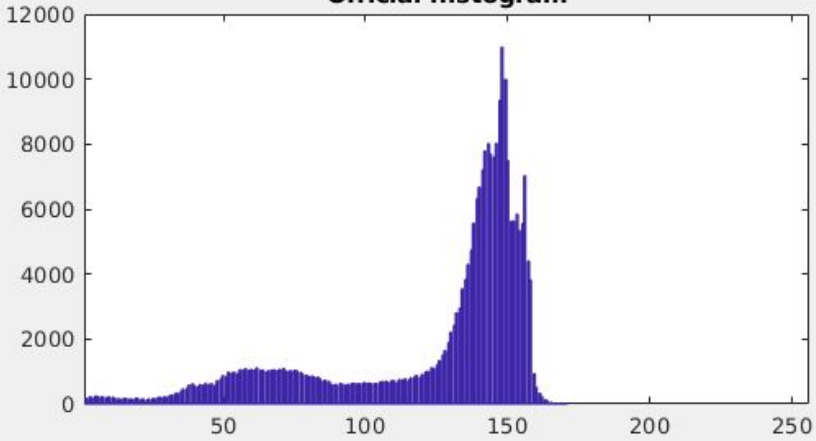
- Create the empty `hist_vector` as an accumulator vector, the number of elements should be the number of possible pixel intensities (256).
- Iterate through your input image (`input_img`) with two (nested) `for` loops, registering the intensity-values of every pixel in your accumulator vector:
  `hist_vector(idx) = hist_vector(idx) + 1;`
  (**Be careful!** Image intensity ∈ [0, 255], Matlab vector index ∈ [1, 256])

The sum of your `hist_vector` should give the total number of pixels present in your image.

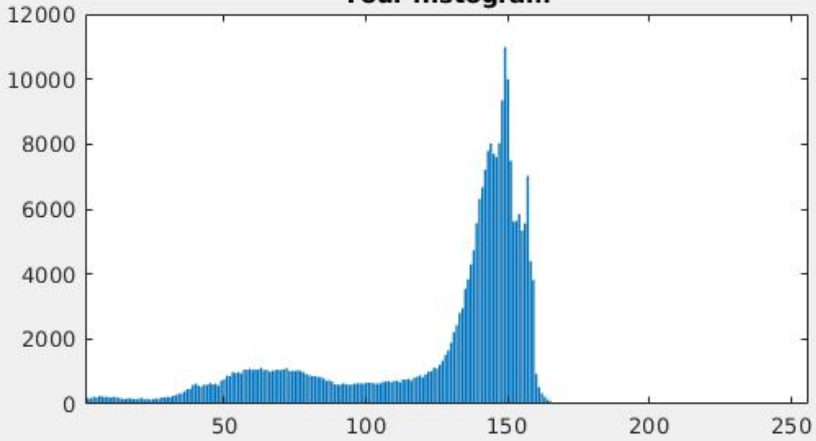
Run `script4.m` which will plot your returned vector as a bar chart.

Grayscale input

Official histogram



Your histogram



Exercise 5

Implement the function `stretch_lin` in which:

- Find the minimum and maximum intensity values of your input image (`input_img`).
- Stretch its dynamic range with the formula given on Slide 41.

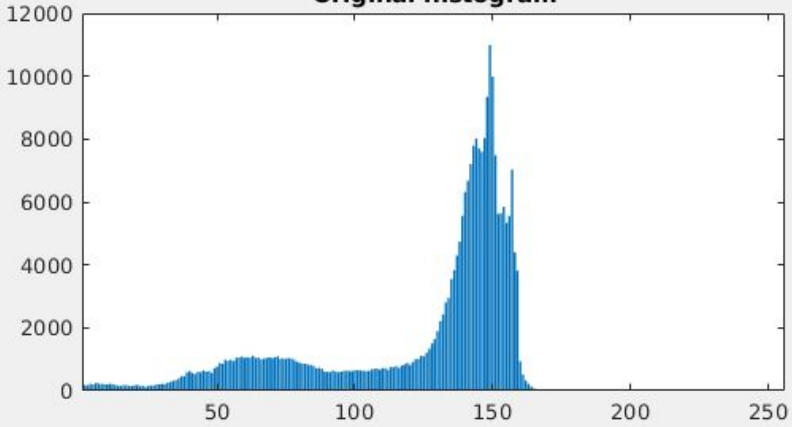
Your resulting image should contain rounded values in the range [0, 255] with type `uint8`.

Run `script5.m` to check your implementation.

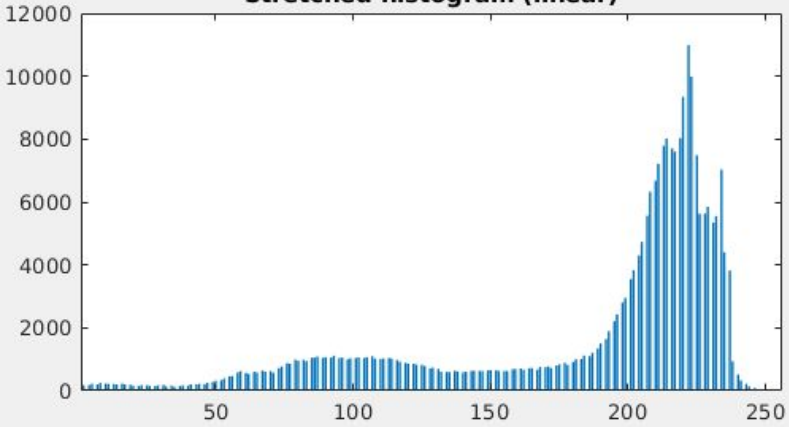
Original image

Stretched image (linear)

Original histogram



Stretched histogram (linear)



Exercise 6

Implement the function `stretch_log` in which:

- Apply the point-wise log transformation at every pixel (as given on Slide 43).
- Find the minimum and maximum intensity values of your transformed image.
- Stretch its dynamic range with the formula given on Slide 41.

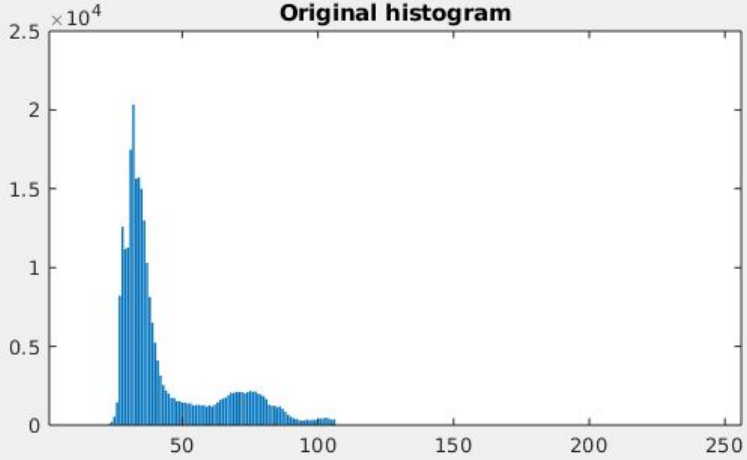
Your resulting image should contain rounded values in the range [0, 255] with type `uint8`.

Run `script6.m` to check your implementation.

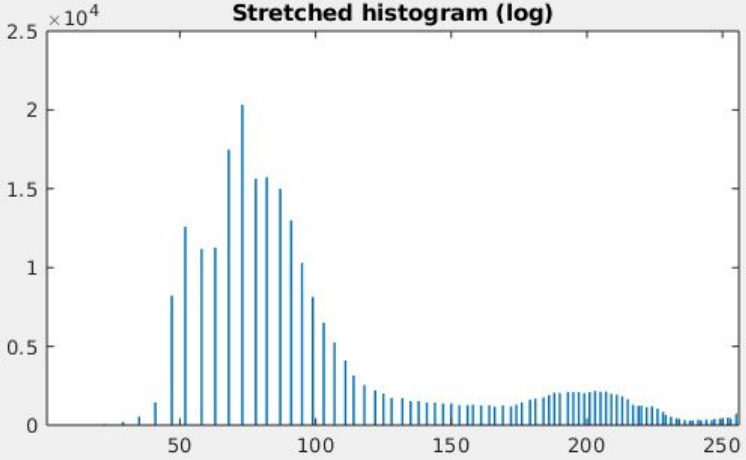
Original image

Stretched image (log)

Original histogram



Stretched histogram (log)



THE END