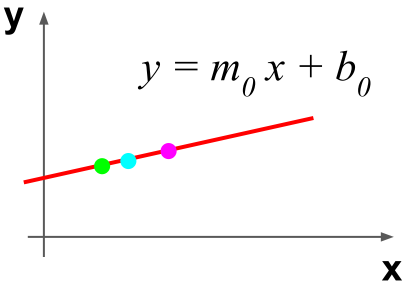
Lab 04

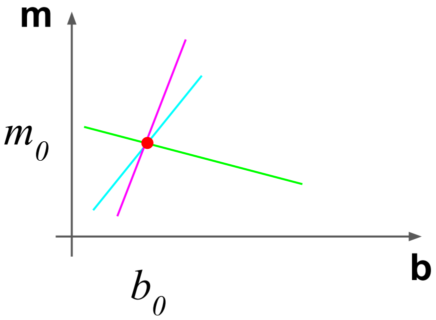
Basic Image Processing
Fall 2024

Hough Transform – Introducing the Hough space

image space

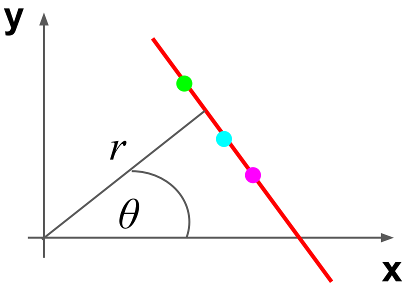


m-b space

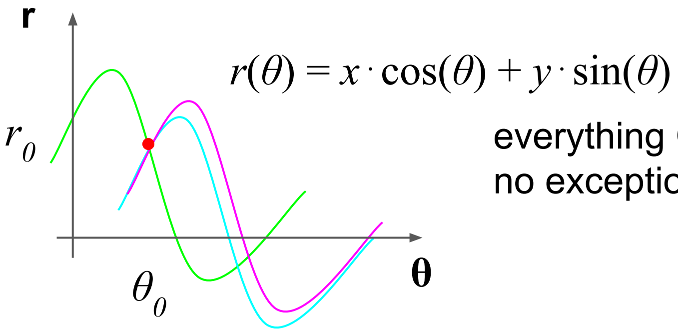


everything OK,
except when m=∞

image space

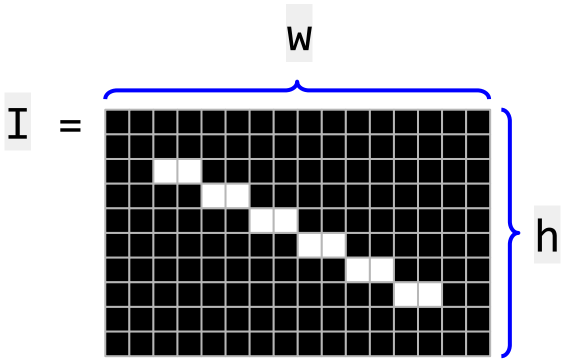


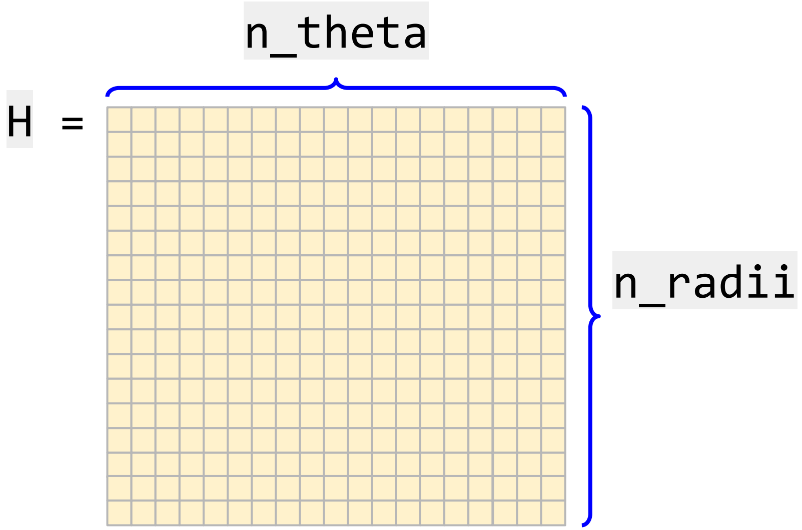
Hough space (r-θ space)



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 twice the value of $r_{\max}$ 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

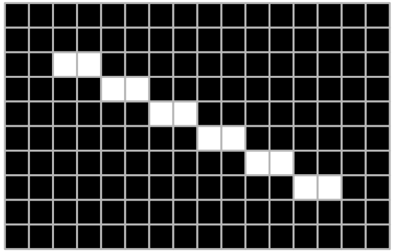
Store the vote (increment `H(r, theta)` by one)

Return

Return the matrix containing the votes.

Example: running of the algorithm

`I` =

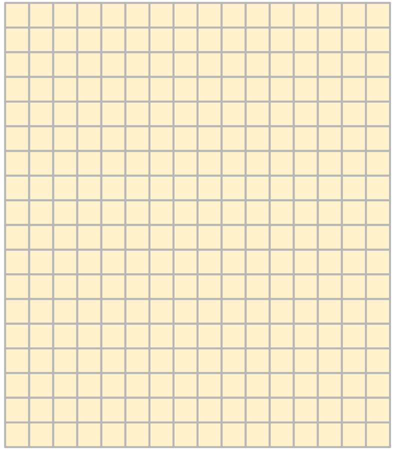


Initialization

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

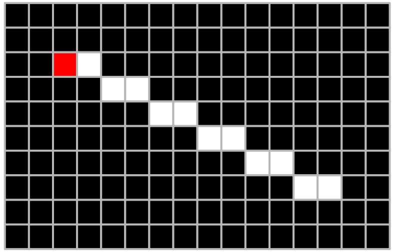
The size of the input image was used to determine the maximal possible radius, which gives the number of rows in H (actually, `2*r_max+1`).

H =



Example: running of the algorithm

`I` =



Initialization

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

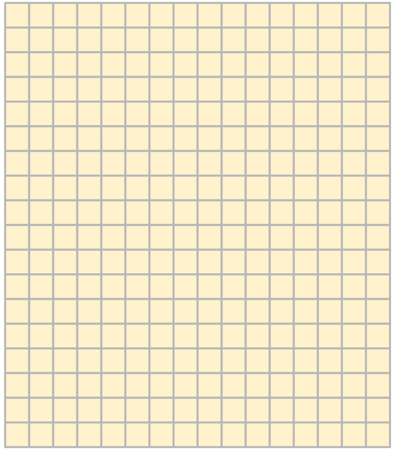
Voting

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

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

`I(3, 3)`

`H` =



Example: running of the algorithm

Initialization

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

Voting

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

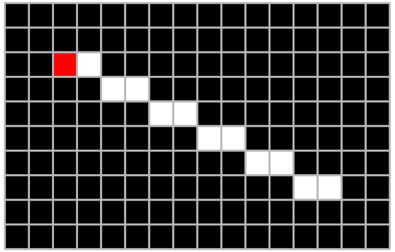
For each `theta` value `theta = 1:180`

`I(3, 3)`

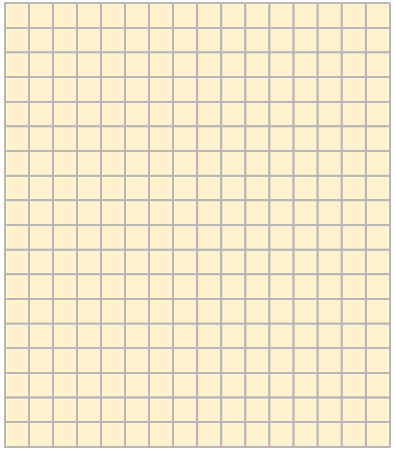
In this first iteration the first θ value is selected; it is

`theta = 1`

`I` =



H =



Example: running of the 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

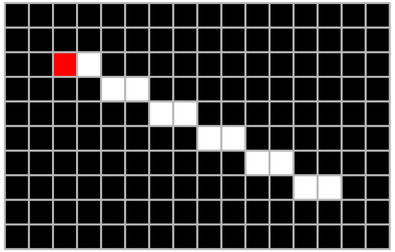
$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

`I(3, 3)`

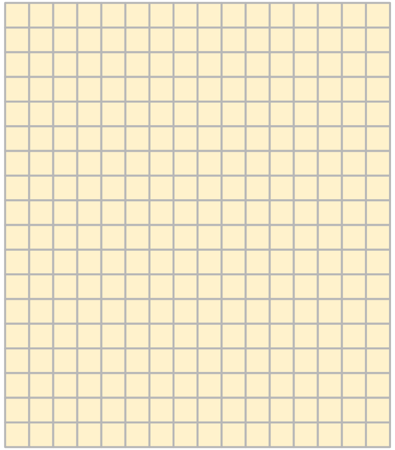
`theta = 1`

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

`I` =

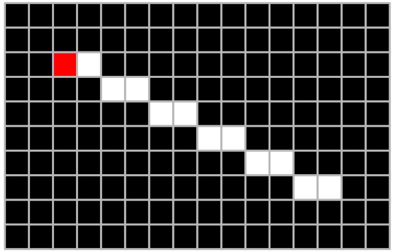


`H` =



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

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

`I(3, 3)`

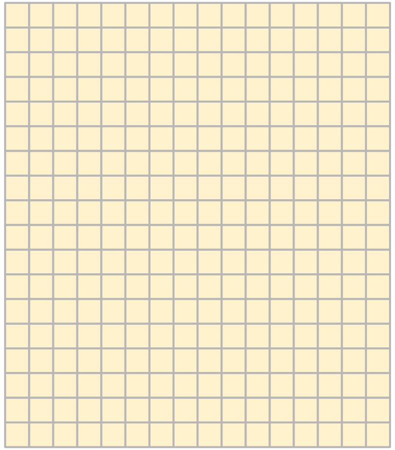
`theta = 1`

$r(1°) = 3.05$

The computed radius value is rounded

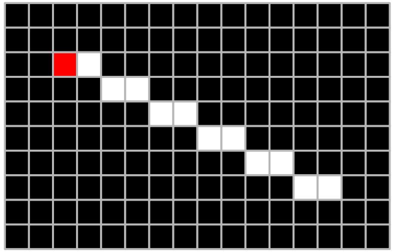
`r = 3`

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

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)

`I(3, 3)`

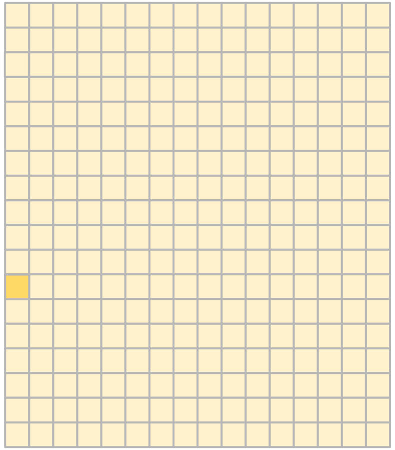
`theta = 1`

$r(1°) = 3.05$

`r = 3`

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

H =



Example: running of the 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)

`I(3, 3)`

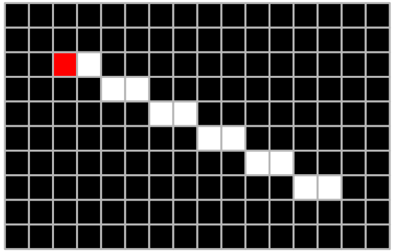
`theta = 1`

$r(1°) = 3.05$

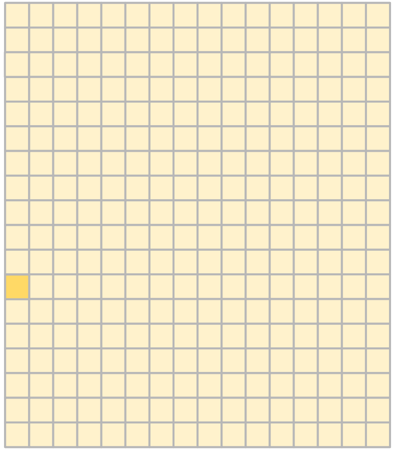
`r = 3`

Innermost loop core is done,
do the next iteration!

`I` =

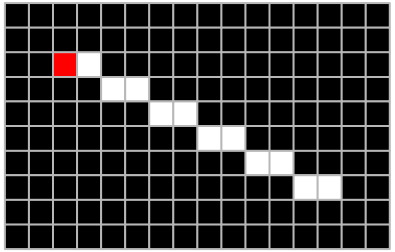


`H` =



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

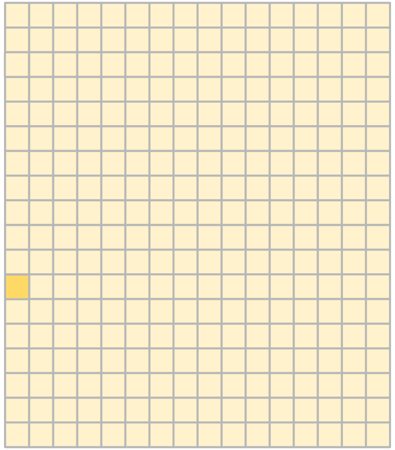
For each `theta` value `theta = 1:180`

`I(3, 3)`

In this <u>second</u> iteration the next θ value is selected; it is

`theta = 2`

`H` =



Example: running of the 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

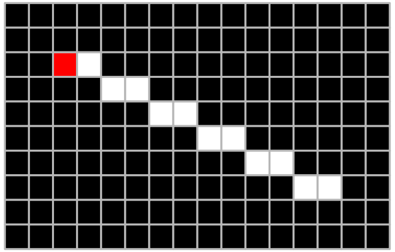
$$r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

`I(3, 3)`

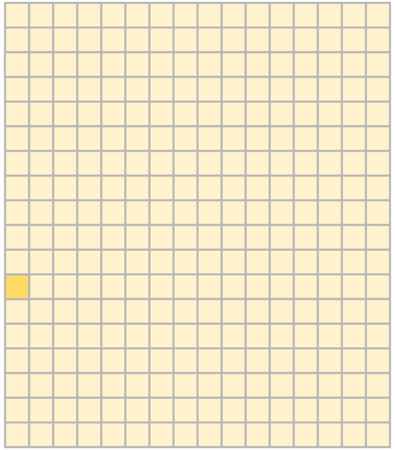
`theta = 2`

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

`I =`

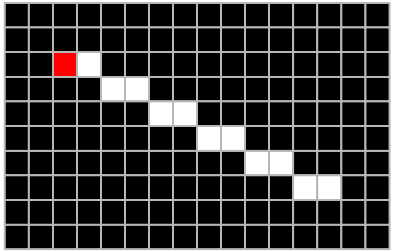


`H =`



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

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

`I(3, 3)`

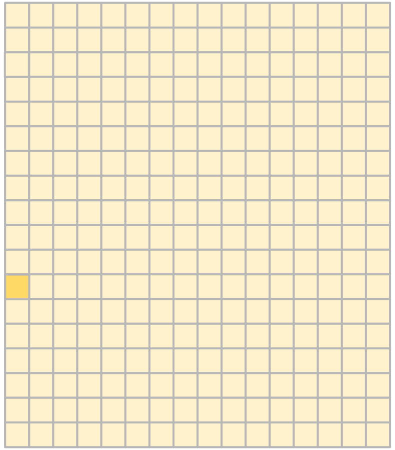
`theta = 2`

$r(1°) = 3.10$

The computed radius value is rounded

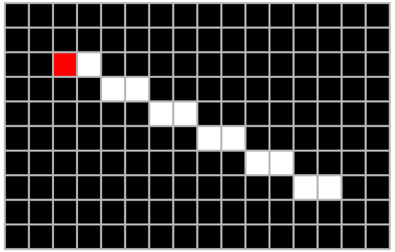
`r = 3`

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

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)

`I(3, 3)`

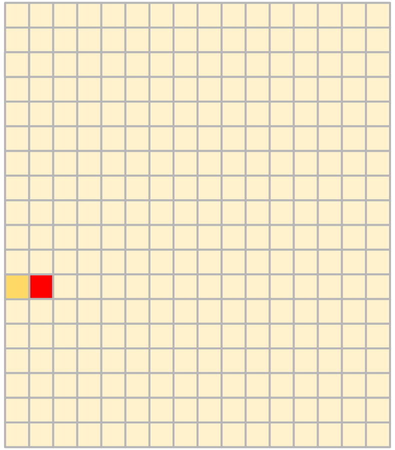
`theta = 2`

$r(1°) = 3.10$

`r = 3`

The element at `H(3 + r_max+1, 2)` is selected and its value is incremented.

`H` =



Example: running of the 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)

`I(3, 3)`

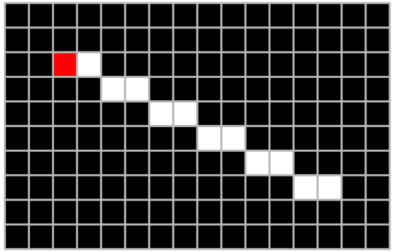
`theta = 2`

$r(1°) = 3.10$

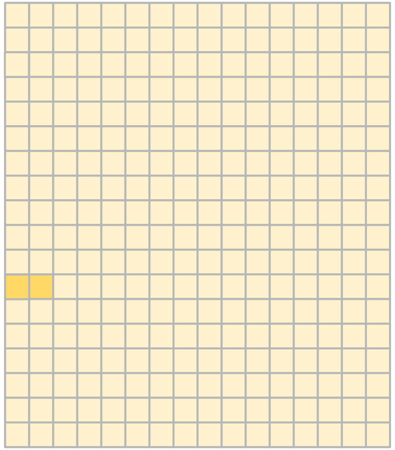
`r = 3`

Innermost loop core is done,
do the next iteration!

`I` =

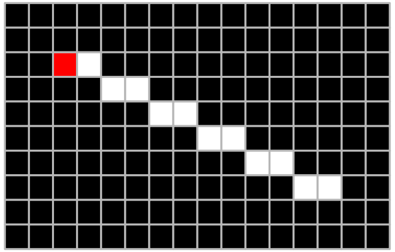


H =



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

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

Calculate the radius using the formula

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

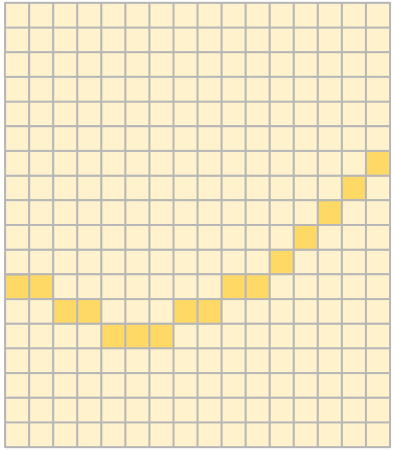
Do the quantization (rounding) of the radius value

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

`I(3, 3)`

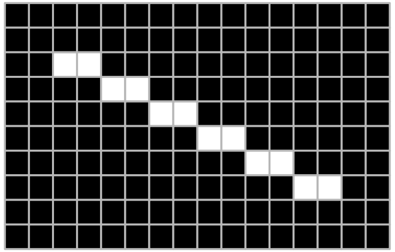
After the completion of all iterations with the `theta` angle the Hough matrix is filled with votes coming from the edge pixel at `I(3, 3)`

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`

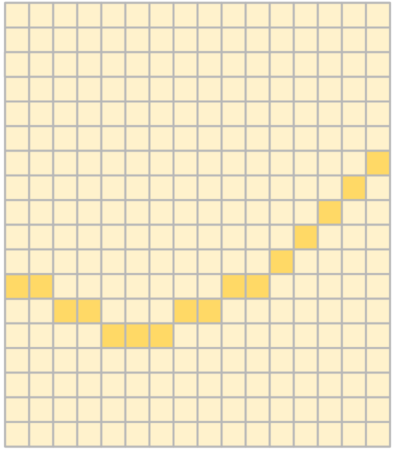
Calculate the radius using the formula

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

Do the quantization (rounding) of the radius value

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

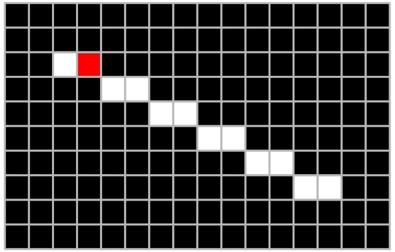
H =



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

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

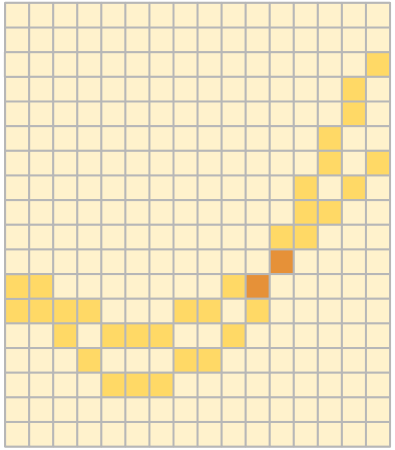
Store the vote (increment `H(r, theta)` by one)

`I(3, 4)`

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.

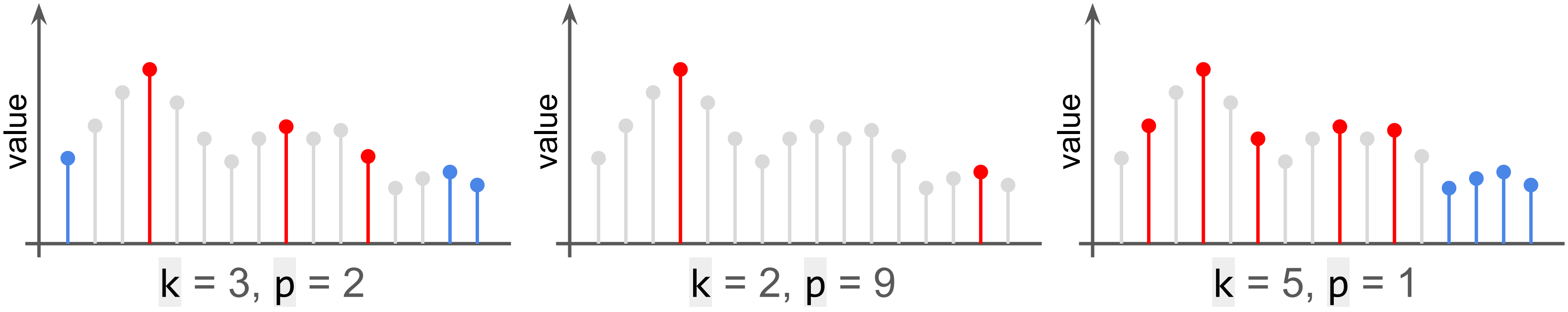
`H` =



Non-maximum suppression – Goal

We use the non-maximum suppression 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`.



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.

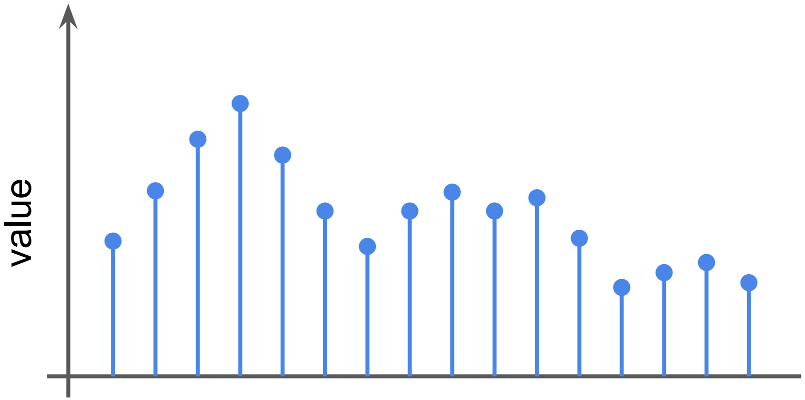
Example: running of the algorithm

Initialization

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

k = 3, p = 2

return:



Example: running of the algorithm

Initialization

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

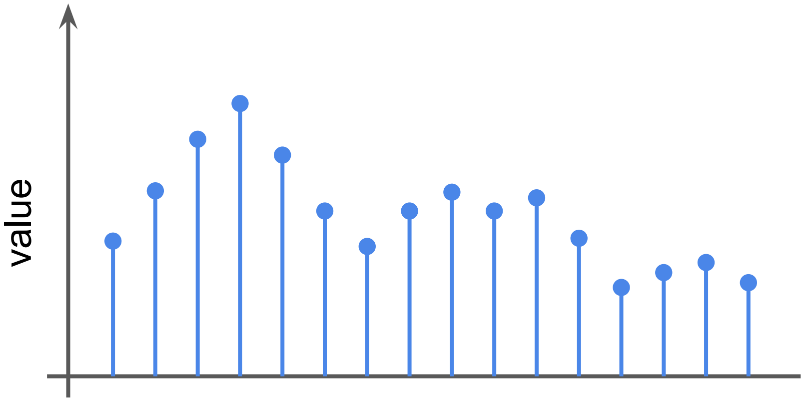
`k` = 3, `p` = 2

Iterative counting

While `k` is not zero

```
k = 3
```

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

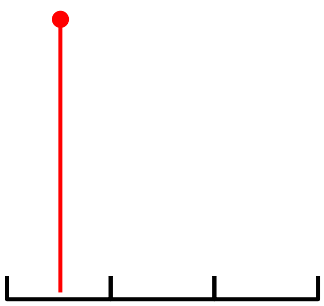
While `k` is not zero

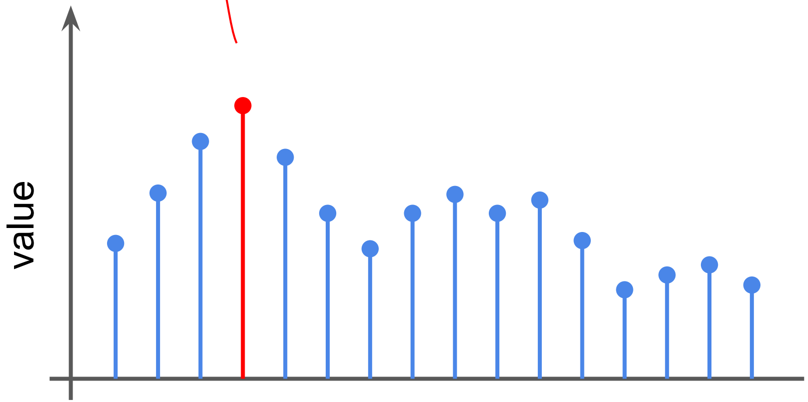
`k = 3`

Find the global maximum in the data

Put this maximum point into the return array

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 = 3

While k is not zero

Find the global maximum in the data

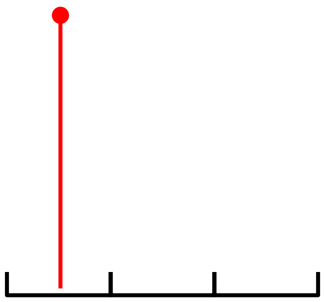
Put this maximum point into the return array

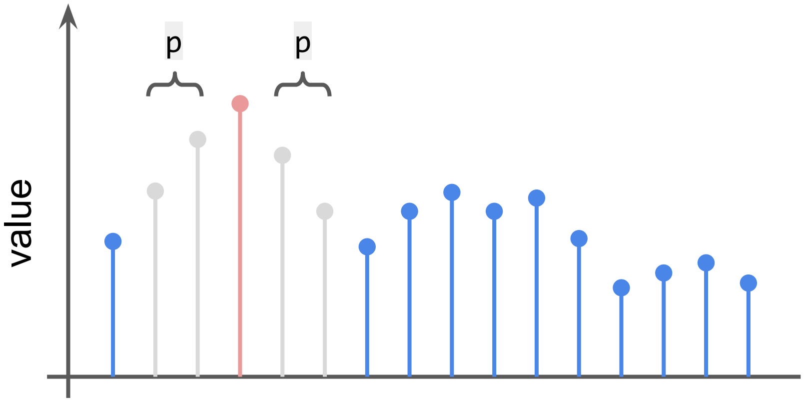
Suppress the maximum point and all the points in its neighborhood with a radius p

Decrease k by 1.

k := 2

return:





Example: running of the algorithm

Initialization

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

`k` = 3, `p` = 2

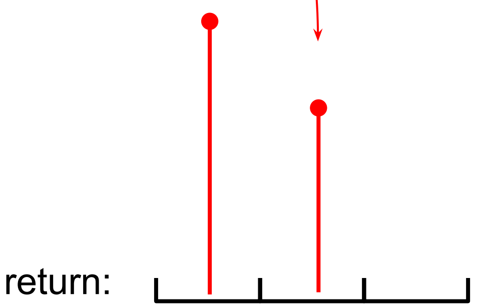
Iterative counting

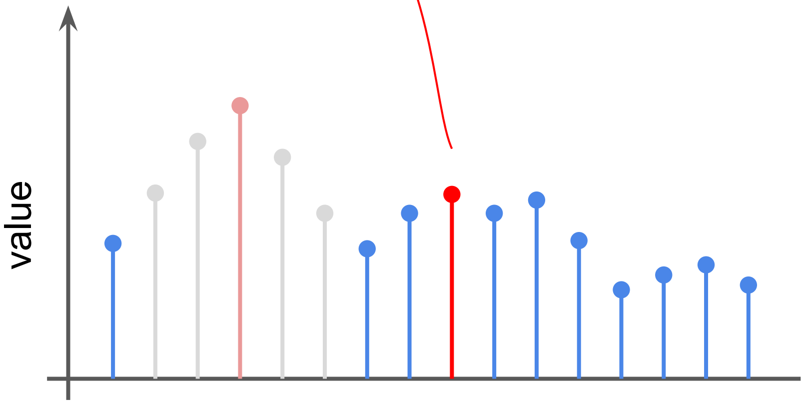
`k = 2`

While `k` is not zero

Find the global maximum in the data

Put this maximum point into the return array





Example: running of the algorithm

$k = 3$, $p = 2$

Initialization

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

Iterative counting

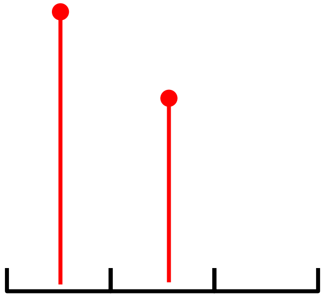
`k = 2`

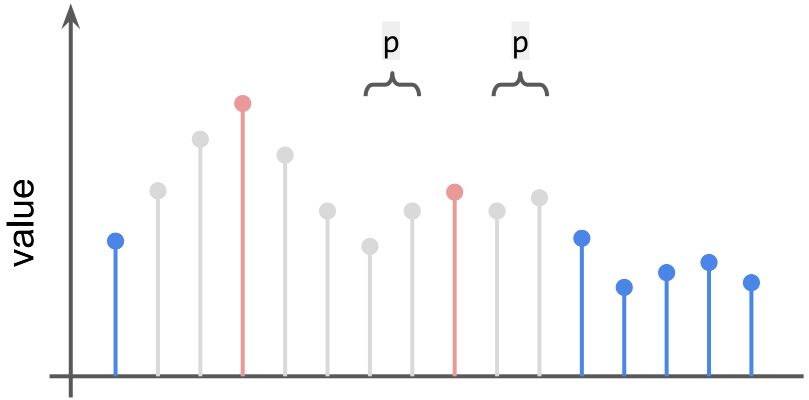
While `k` is not zero

- 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).

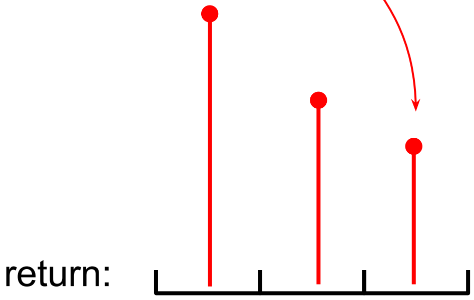
`k` = 3, `p` = 2

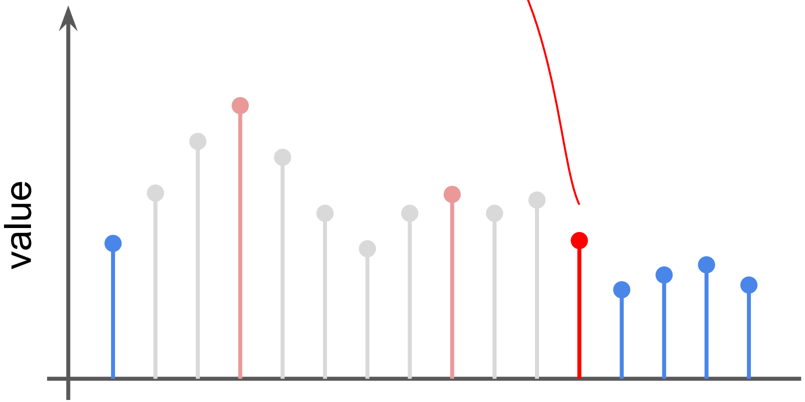
Iterative counting

While `k` is not zero

`k = 1`

- 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

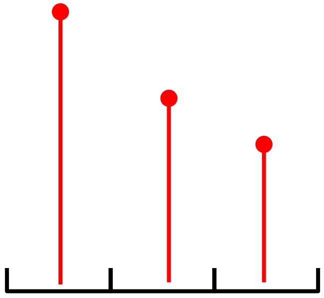
Put this maximum point into the return array

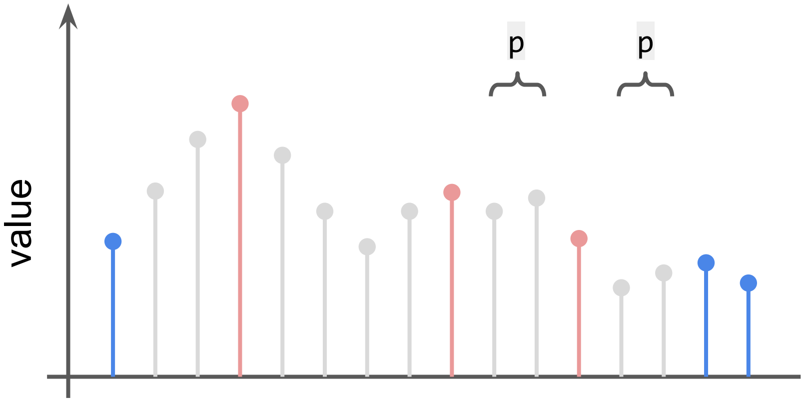
Suppress the maximum point and all the points in its neighborhood with a radius `p`

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

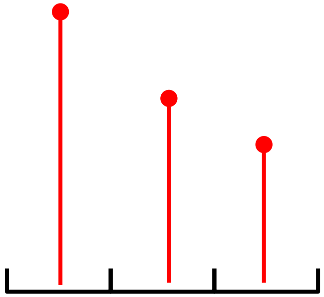
    Suppress the maximum point and all the points in its neighborhood with a radius p

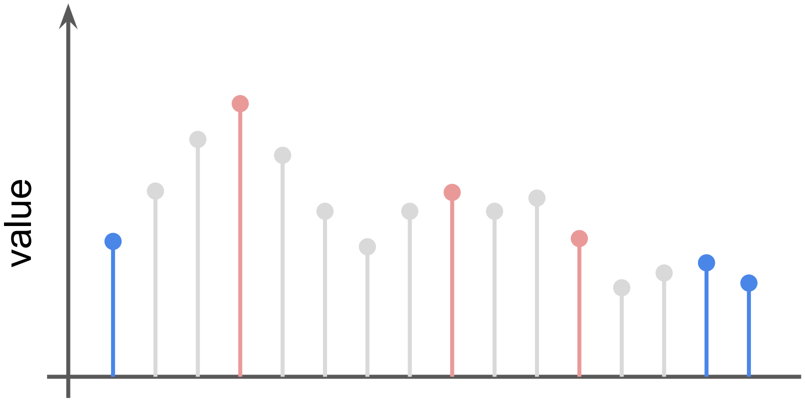
    Decrease k by 1.

Return

Return the array containing the maxima points.

return:





Now please

download the ‘Lab 04’ code package

from the

moodle

Exercise 1

Implement the function `my_hough` in which:

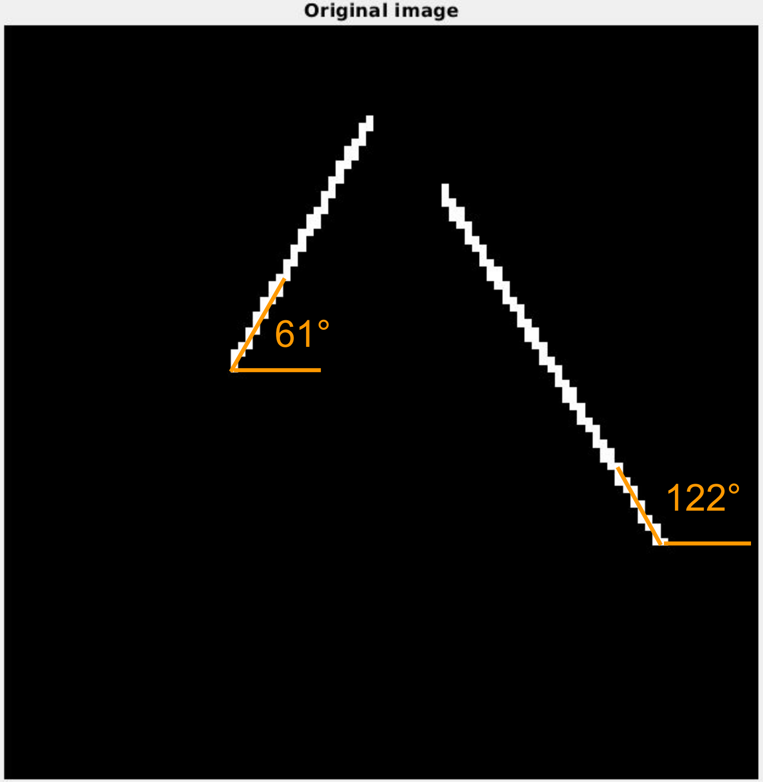
Realize the Hough Transform algorithm as described on Slide 4.

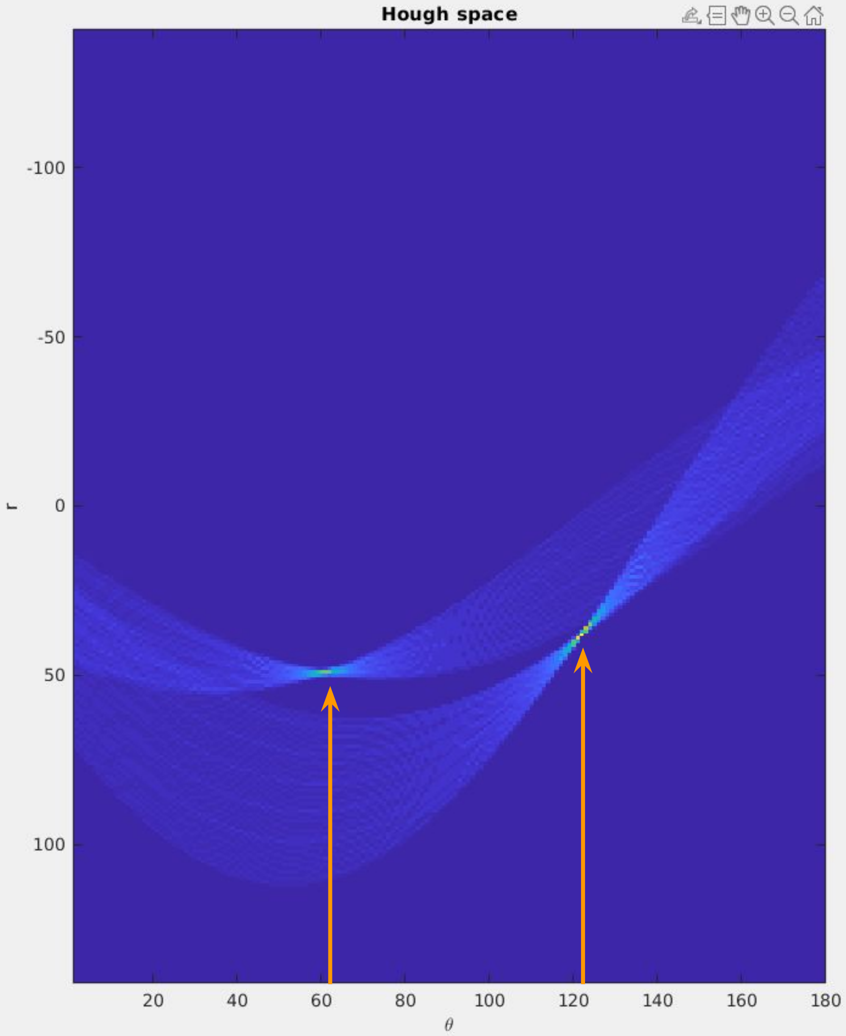
- Initialize the `H` matrix, where
  - the number of rows is twice the longest possible `r` radius +1 on your original image (diagonal)
  - the number of columns is 180, referring the range of the angle theta $\in$ [1, 180]
- Iterate through your input image (`input_img`) with two (nested) `for` loops, 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 black-and-white, the matrix contains {0,1} values only.

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:

- `A`: 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 (`A`), 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` = …

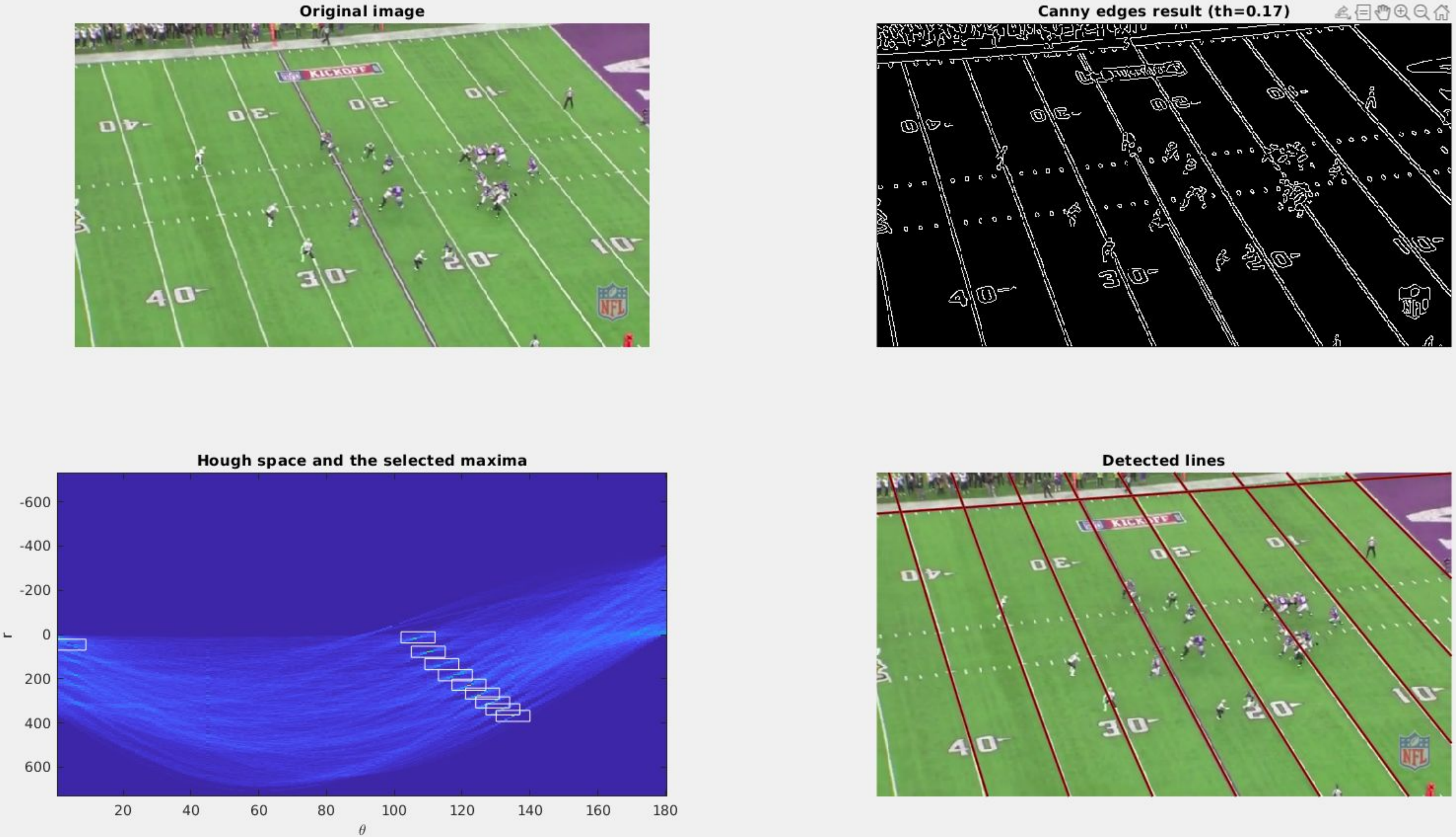
`y_2` = …

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
KICKOFF
Canny edges result (th=0.17)
Hough space and the selected maxima
r
θ
-600
-400
-200
0
200
400
600
20
40
60
80
100
120
140
160
180
Detected lines

THE END