Lab 04

Basic Image Processing
Fall 2018

The Histogram of an Image

- **Histogram**:
  $h(k)$ = the number of pixels on the image with value k.

Original Image*

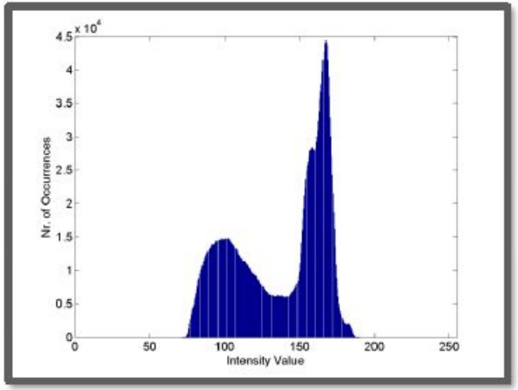


Image Histogram

- 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:

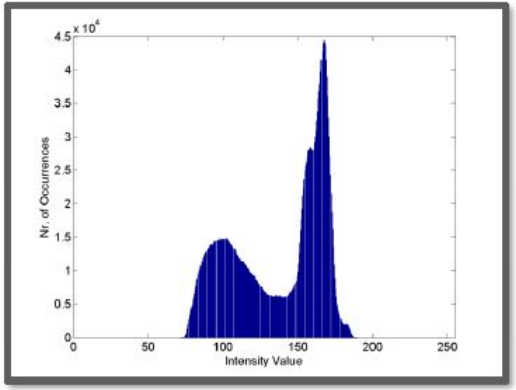


Image Histogram

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

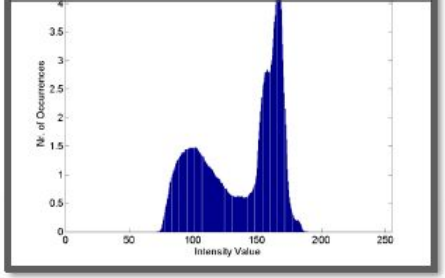
$$x_{\max} = \max_{n_1, n_2}(x(n_1, n_2)) \qquad x_{\min} = \min_{n_1, n_2}(x(n_1, n_2))$$

Histogram Transformations

- **Histogram Stretching:**

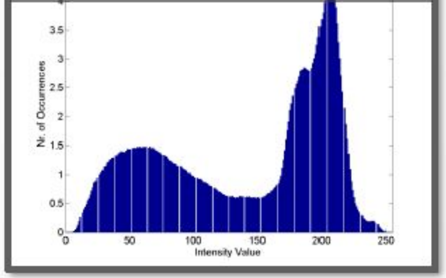
Original image

Original histogram



Histogram stretched image

Stretched histogram



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)

Now please

download the ‘Lab 04’ code package

from the

submission system

Part1

- `calc_hist_vector`
  calculates the histogram vector of its input image
- `stretch_lin`
  stretches the histogram (image intensities) to the maximal range, linearly
- `stretch_log`
  applies point-wise, logarithmic intensity transformation on the image, then stretches it

Part2

- `my_hough`
  transforms the input image to the Hough-space
- `non_max_sup`
  realizes non-maxima suppression on the given 2D data

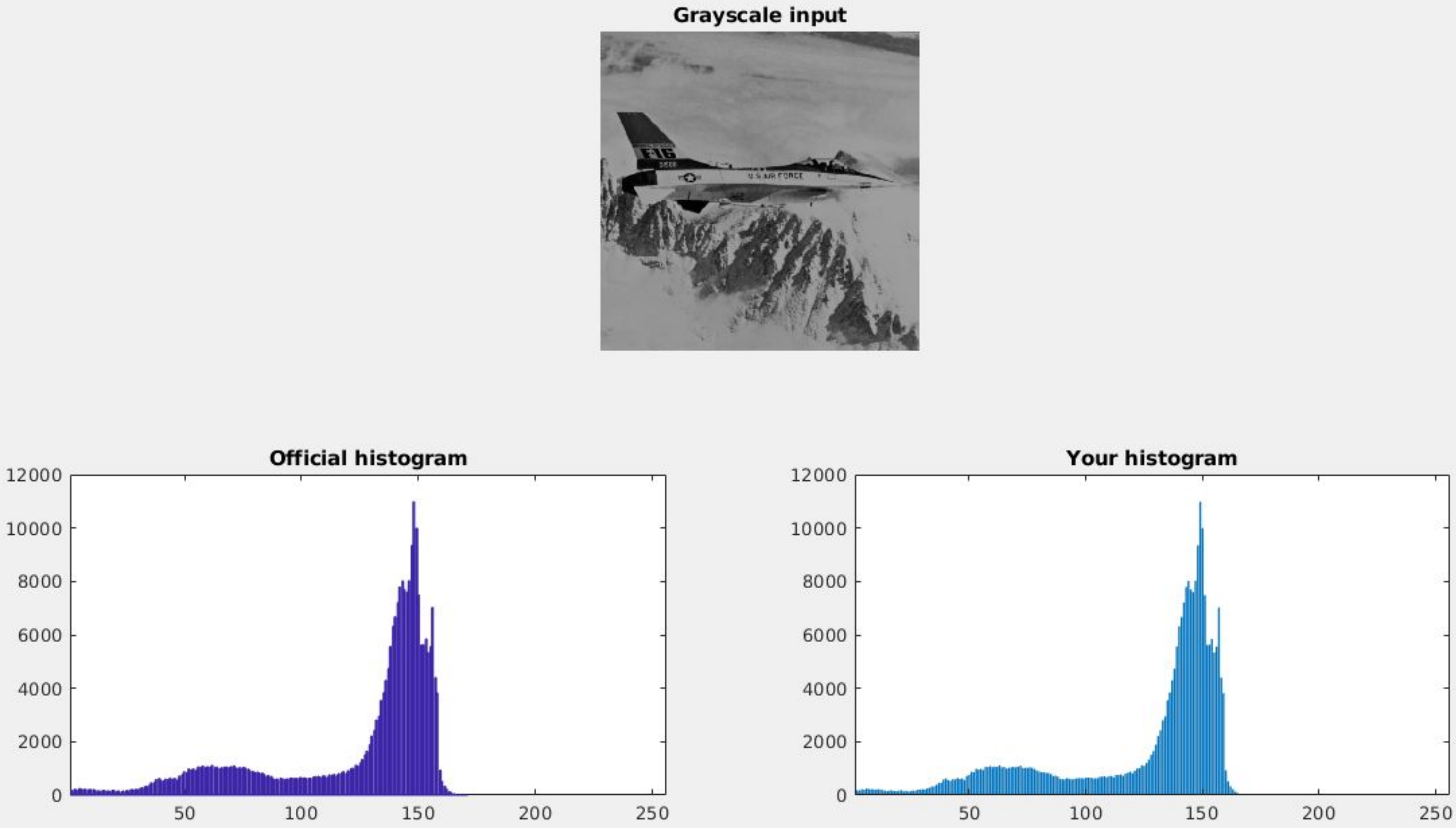
Exercise 1

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;`
  (**NB:** image intensity ∈ [0, 255], matlab vector index ∈ [1, 256].)

The summation of your `hist_vector` should result in to the total number of pixels present in your image.

Run `script1.m` which will plot your returned vector as a bar chart.


Grayscale input
Official histogram
Your histogram
12000
10000
8000
6000
4000
2000
0
50
100
150
200
250

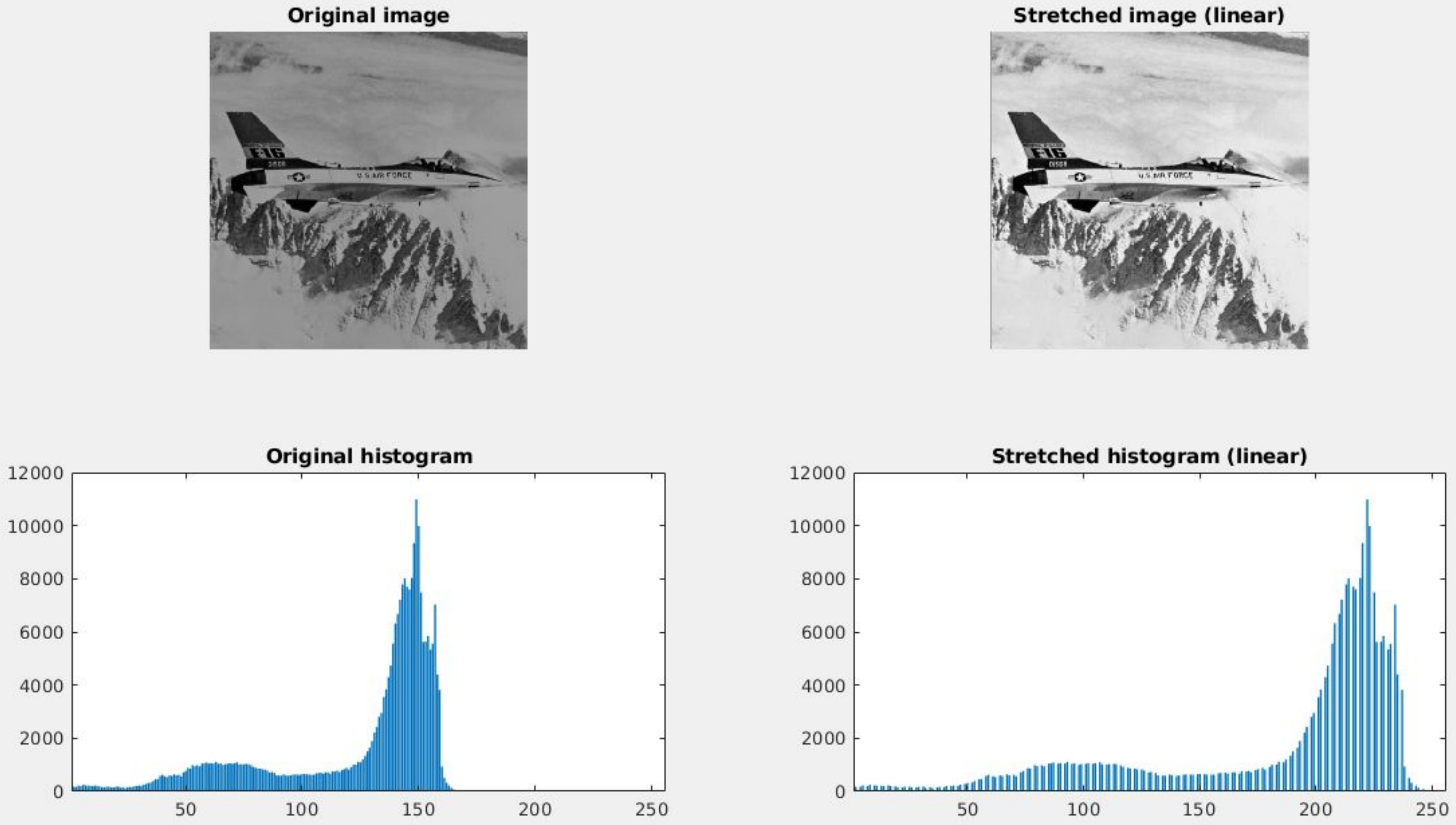
Exercise 2

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 3.

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

Run `script2.m` to check your implementation.


Original image
Stretched image (linear)
Original histogram
12000
10000
8000
6000
4000
2000
0
50
100
150
200
250
Stretched histogram (linear)
12000
10000
8000
6000
4000
2000
0
50
100
150
200
250

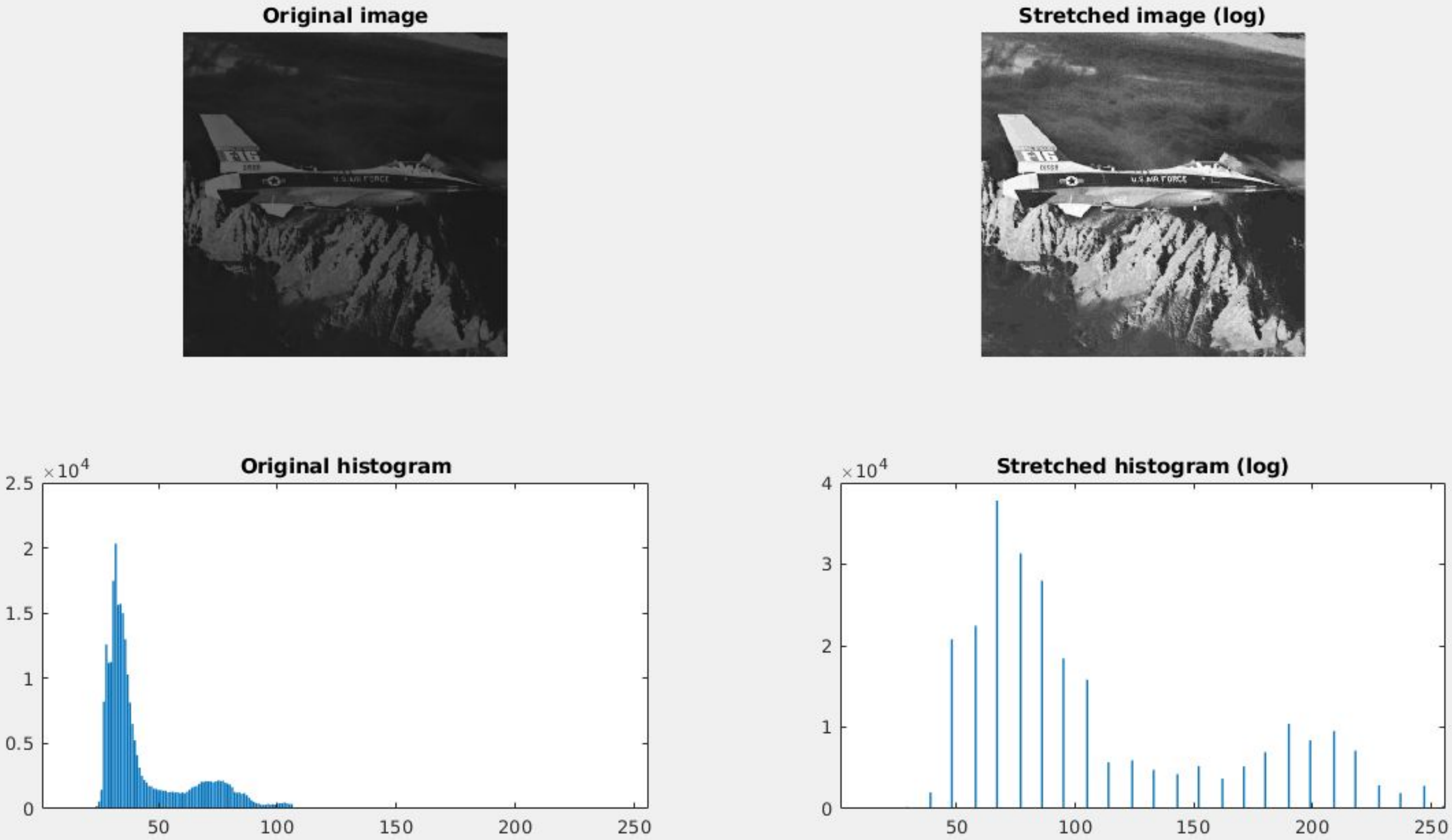
Exercise 3

Implement the function `stretch_log` in which:

- Iterate through your input image (`input_img`) with two (nested) `for` loops, applying point-wise log transformation at every pixel (as given on Slide 5).
- Find the minimum and maximum intensity values of your transformed image.
- Stretch its dynamic range with the formula given on Slide 3.

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

Run `script3.m` which will


Original image
Stretched image (log)
Original histogram
×10^4
2.5
2
1.5
1
0.5
0
50
100
150
200
250
Stretched histogram (log)
×10^4
4
3
2
1
0
50
100
150
200
250

Part1

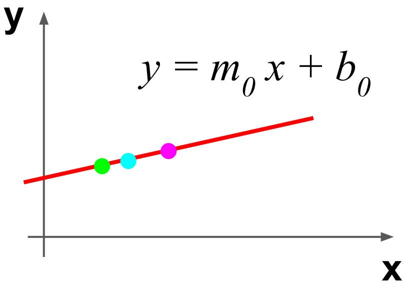
- `calc_hist_vector`
  calculates the histogram vector of its input image
- `stretch_lin`
  stretches the histogram (image intensities) to the maximal range, linearly
- `stretch_log`
  applies point-wise, logarithmic intensity transformation on the image, then stretches it

Part2

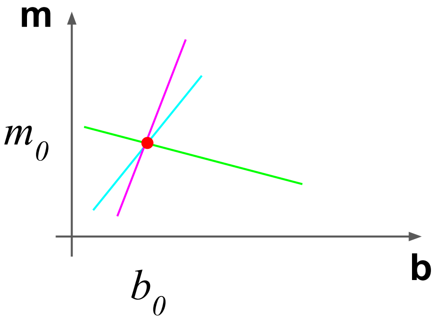
- `my_hough`
  transforms the input image to the Hough-space
- `non_max_sup`
  realizes non-maxima suppression on the given 2D data

Introducing the Hough space

image space

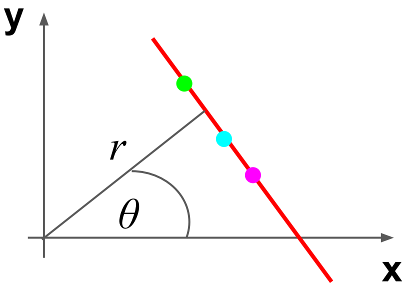


m-b space

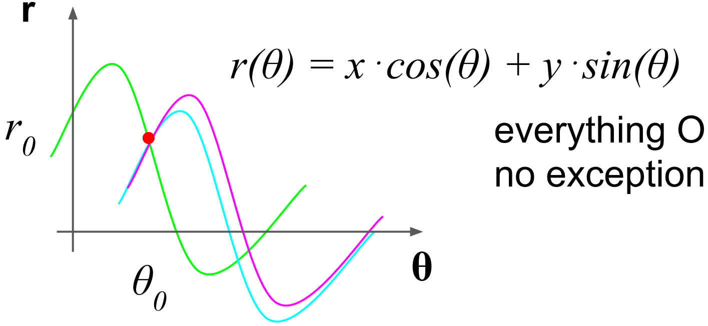


everything OK,
except when m=∞

image space



Hough space (r-θ space)



everything OK,
no exception

Exercise 4

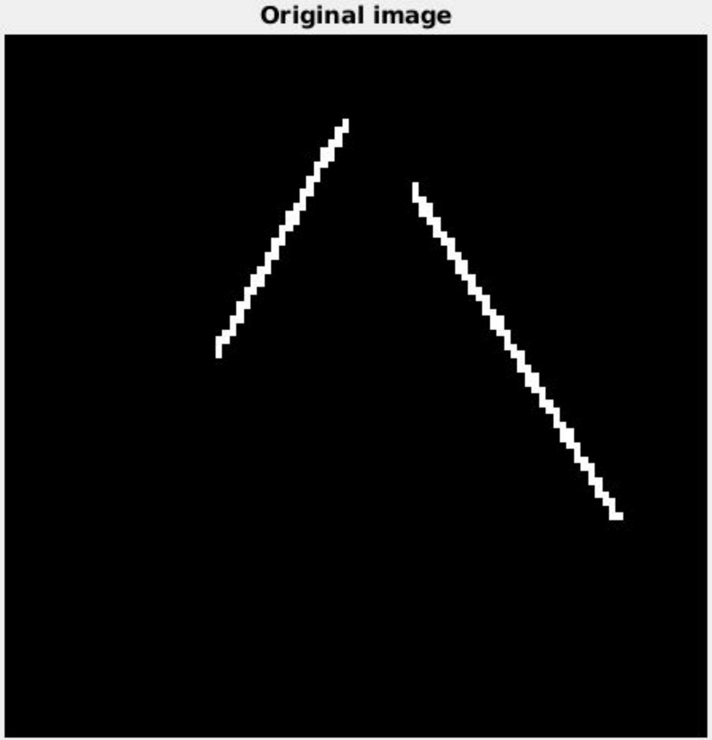
Implement the function `my_hough` in which:

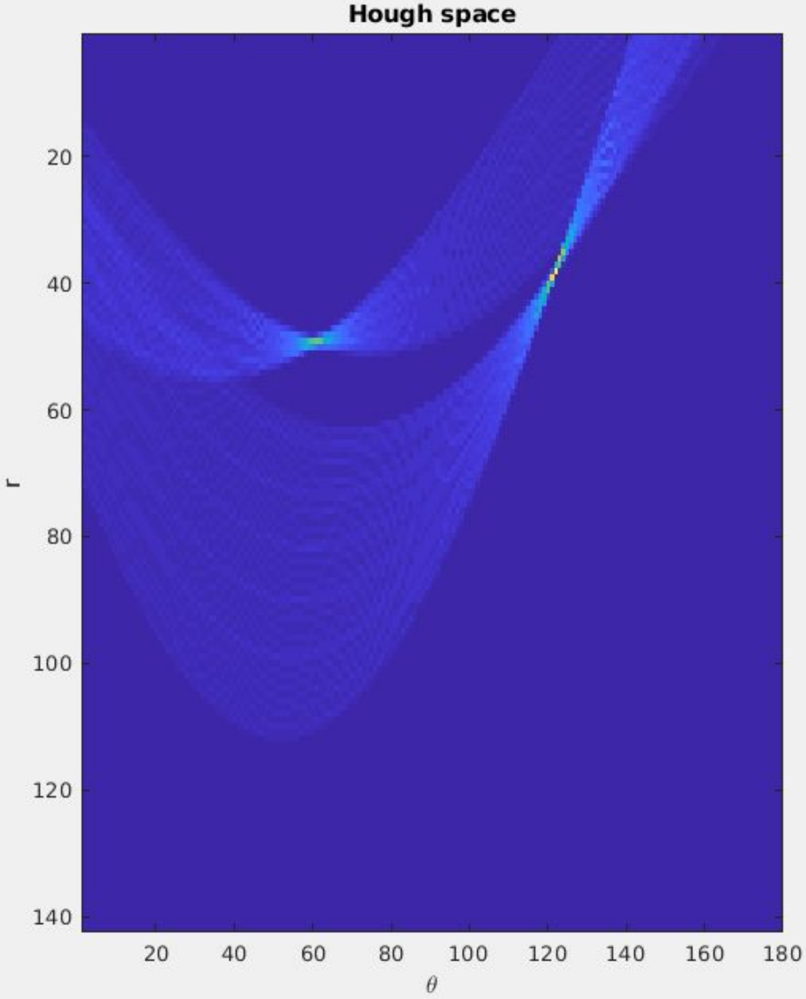
- Create the zero-matrix `H` where
  - the number of rows is the longest possible `r` radius on your original image (it will be the length of the diagonal)
  - the number of columns is 180, referring the dynamic range of the angle `θ` $\in [1, 180]$

  this `H` will be the accumulator for your (r, θ)-occurrences calculated from the image-space.
- Iterate through your input image (`input_img`) with two (nested) `for` loops, calculating the bottom right formula on Slide 15 at every nonzero pixel with all the possible theta values (`θ` $\in [1, 180]$).

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 `script4.m`


Original image


Hough space
20
40
60
80
100
120
140
r
20
40
60
80
100
120
140
160
180
θ

Exercise 5

The **function** `non_max_sup` will have 3 input parameters:

- `H`: input matrix
- `k`: number of maximal values, whose neighboring regions should be suppressed,
- `s`: the radius of the region around a maximal value 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 `θ` index in `r_vect` and `t_vect` arrays,
- zero out the [`-s`, `s`] neighborhood of the maximum point,
- decrease `k`.

Exercise 5 - 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 max.

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

if `H(x_n, y_n)` is maximal

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

where

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

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

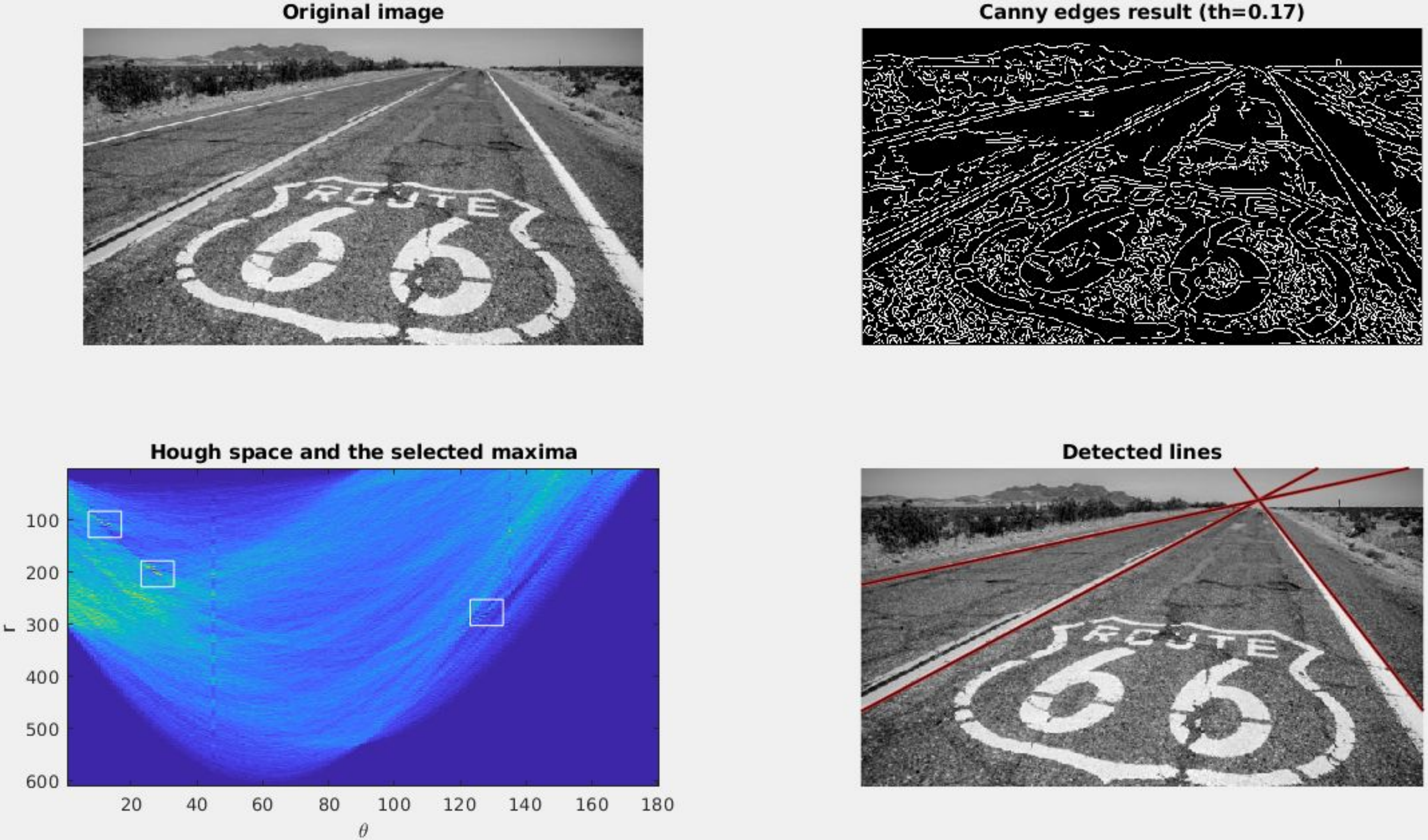
`y_1` = …

`y_2` = …

Exercise 6

After implementing the maximum suppression function, test it using `script5.m`.

Finally, please run `script6.m` to see a practical use of the Hough transform.



Original image
Canny edges result (th=0.17)
Hough space and the selected maxima
Detected lines
r
100
200
300
400
500
600
20
40
60
80
100
120
140
160
180
θ