# Task 1 – Hough Transform

In order to make the explanation simpler and more understandable, we will expand the ‘imshow’ function, as in the MATLAB example for the Hough function. Thus, we obtain:

* 1. Black image with dot in LEFT TOP corner returns the Hough map:



As we can understand, the point is placed on the origin of the place, from which all the values are being calculated. For all the values, we obtain than the line, which originates in the top left corner and is perpendicular to any line crossing the (1,1) dot has a length of 0.

* 1. Black image with dots in (1,1) and (101,1) results in a Hough Map:



From this image we can see that there is 1 line which connects those 2 dots, and this line has a parameters: ; which gives us an understanding as to where the 0 angle of is placed – it is the angle to the line, which goes from the origin (1,1) straight down. The graph is indeed correct, since the line, which has those parameters, passes through the (101,1) point, and is perpendicular to the origin.

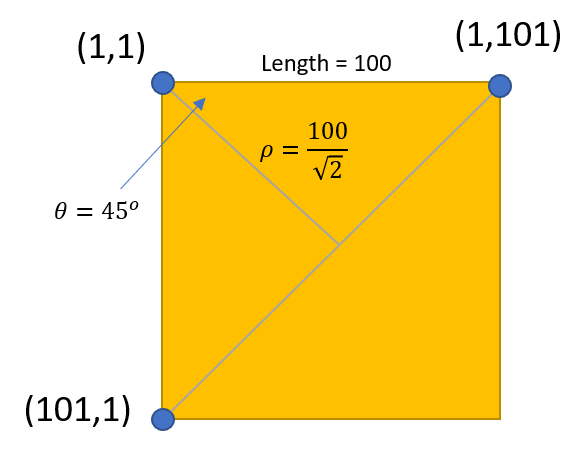
* 1. Adding the third dot at (1,101) gives us



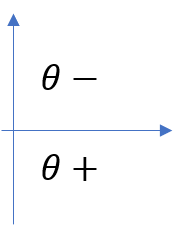
We can observe 2 things:

1. Is has intersection with a Hough line of the second dot precisely at:

Visualizing it:



Now we also know the positive direction:



1. We can see 2 intersections with the straight line (which corresponds to the (1,1) dot), at parameters: and ; Those 2 sets of parameters describe the same line which passes through both of the points (which connects (1,1) and (1,101)).
   1. Adding the dot at (101,101) gives us:



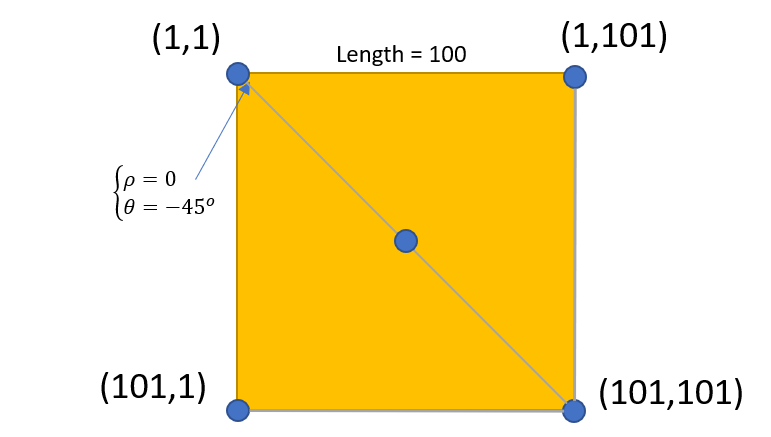
We can identify the line by its biggest value of , since it is the largest distance from the origin (1,1). We can observe 4 intersections, describing 3 lines which connect dots:

1. With (1,1) : intersection at - is 0 because the line crossing the origin has a distance of 0 from the origin. is negative 45 degrees because the perpendicular line will be placed in the upper plane (the region, outside of the borders of the image, to the top, where the values are negative).
2. With (101,1): intersection at and intersection at - it has 2 intersections, since the loop goes over degreees. So, both first and the last line drawn will correspond to the same line (in every case). Since, in our case, both dots (101,1) and (101,101) lie on that same line – they get 2 intersections!
3. With (1,101): intersection at – the perpendicular line to the line, which connects those 2 points, is at and has a length of 100.
   1. Adding the last dot at (51,51) gives:



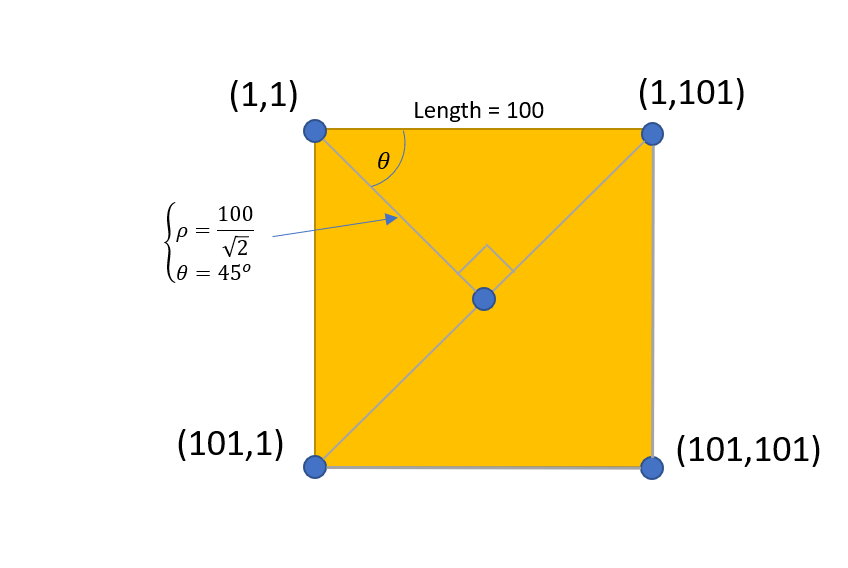
It is easy to identify the line, since its biggest value is 50. It has 2 major intersections, or more precisely, each one contains 3 lines:

1. At parameters with dots (1,1), (101,101). Visualizing the case:



As we can see, indeed, the intersection with those parameters describes the line which connects those 3 dots.

1. At parameters with dots (101,1), (1,101). Visualizing the case:



As we can see, indeed, the intersection with those parameters describes the line which connects those 3 dots.

## 

Creating the binary image of a square (we used padding) results in:



## 

Finding the edges using the Canny edge detection results in:



## 

The Hough transform of the image looks in the following way:



Since we can hardly see anything here (any intersections), we use the proposed MATLAB functions rescale & imadjust, which will increase the contrast of the image. The result is the following:



We can indeed spot 4 lines in the image, which correspond to the following parameters (intersections):

1. , which is the left border of the square
2. , which is the right border of the square
3. & , which BOTH correspond to the upper border of the square
4. & , which BOTH correspond to the lower border of the square

Explanation on why lower and upper border create 2 intersections each: because the initial and the final line which are generated for any point on it is the same exact line, and is the line on which they lay! Same case was with the line in the (a) subsection of the question.

## 

The lines are represented as the maximum point of the Hough transform. This is true, because it shows, that a certain line, describes by some certain parameters , passes through many points. Which means that those points lay on the same line.

MATLAB provides the function *houghpeaks.*  From the MATLAB helpfile:

[peaks](https://www.mathworks.com/help/images/ref/houghpeaks.html#buw9d0p-1-peaks) = houghpeaks([H](https://www.mathworks.com/help/images/ref/houghpeaks.html" \l "buw9d0p-1-H),[numpeaks](https://www.mathworks.com/help/images/ref/houghpeaks.html#buw9d0p-1-numpeaks)) locates peaks in the Hough transform matrix, H, generated by the [hough](https://www.mathworks.com/help/images/ref/hough.html) function. numpeaks specifies the maximum number of peaks to identify. The function returns peaks a matrix that holds the row and column coordinates of the peaks.

Consult the help file for houghpeaks for details????

## 

Using the function *hougheaks* on our image results in the following output (after specifying number of peaks > 4 ):

|  |  |
| --- | --- |
| 933 | 1 |
| 1333 | 901 |
| 531 | 1 |
| 1735 | 901 |

Those are the indexes of the values in the Hough matrix. Finding the exact values of those parameters through those indexes results in:

|  |  |
| --- | --- |
|  |  |
| -100 | -90 |
| 100 | 0 |
| -301 | -90 |
| 301 | 0 |