Project 1

Steps

1.Gray the image

2.Blur the Image using GaussianBlur

3.Detect Edges using Canny edge detection

4.Mask out most of the image and only keep the bottom part of the road in view

5.Hough Transform to convert the pixel dots that were detected as edges into meaningful lines.

6.Separate the lines as right lane or left lane

7. Extroplate

8.remove outliers

9.draw lines detected by Hough Transform

10.Render image on top of original

11.Render Video

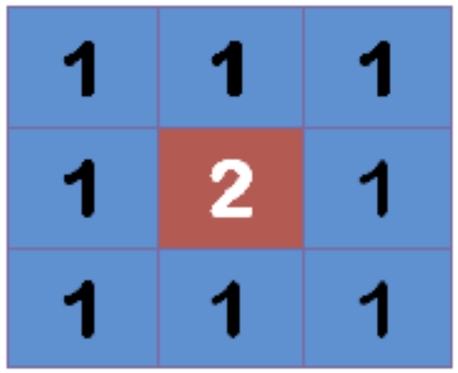
Short Comings

- During steep turns the lines are not perfect

Algorithm Internals

Gaussian Blur theory

A **Gaussian blur** (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function.



On the above graph, 2 is the center point, the surrounding points are 1.

The center point will take the average value of its surrounding points, it will be 1. The center point will lose its detail.

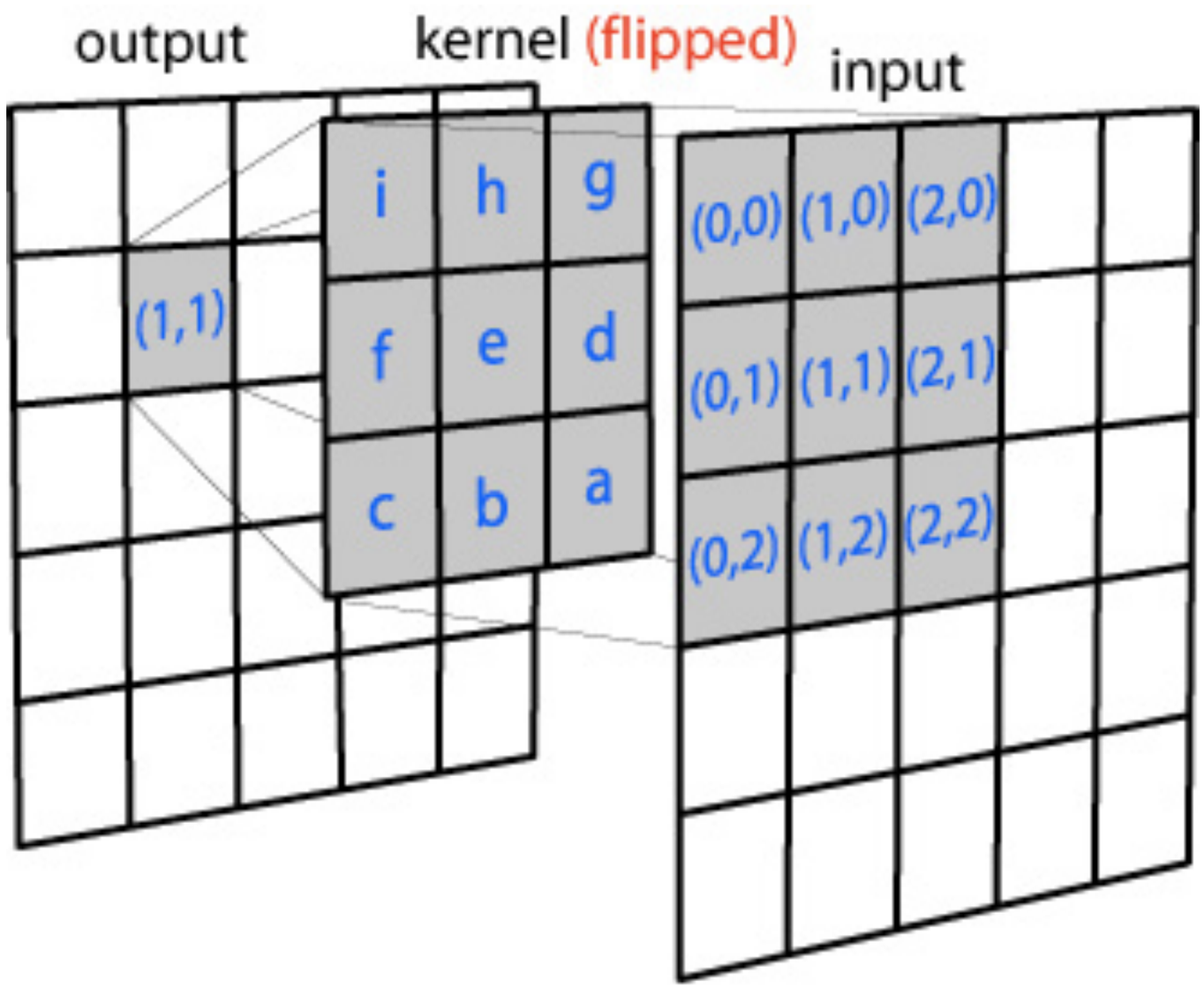
- Imagine you have an image defined as follow

- 00 01 02 03 04 05 06 07 08 09
- 10 11 12 13 14 15 16 17 18 19
- 20 21 22 23 24 25 26 27 28 29
- 30 31 32 33 34 35 36 37 38 39
- 40 41 42 43 44 45 46 47 48 49
- 50 51 52 53 54 55 56 57 58 59
- 60 61 62 63 64 65 66 67 68 69
- 70 71 72 73 74 75 76 77 78 79
- 80 81 82 83 84 85 86 87 88 89
- 90 91 92 93 94 95 96 97 98 99

- To apply the gaussian blur you would do the following:

- For pixel 11 you would need to load pixels 0, 1, 2, 10, 11, 12, 20, 21, 22.
- you would then multiply pixel 0 by the upper left portion of the 3x3 blur filter. Pixel 1 by the top middle, pixel 2, pixel 3 by top right, pixel 10 by middle left and so on.Then add them altogether and write the result to pixel 11. As you can see Pixel 11 is now the average of itself and the surrounding pixels.

The above are graphs of original, 3 pixels blur radius and 10 pixels blur radius. The bigger the blur radius, the more blur the picture is.



The rectangular group of pixels, surrounding the pixel to filter is called the *kernel*. The Kernel moves along with the pixel that is being filtered. So that the filter pixel is always in the center of the kernel, the width and height of the kernel must be odd. you would multiply the value at (0,0) in the input array by the value at (i) in the kernel array, the value at (1,0) in the input array by the value at (h) in the kernel array, and so on. and then add all these values to get the value for (1,1) at the output image.

Image

14	15	16
24	25	26
34	35	36

Kernel with weights

0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

Each point multiplies its weight value:

14x0.0947416	15x0.118318	16x0.0947416
24x0.118318	25x0.147761	26x0.118318
34x0.0947416	35x0.118318	36x0.0947416

Now we have

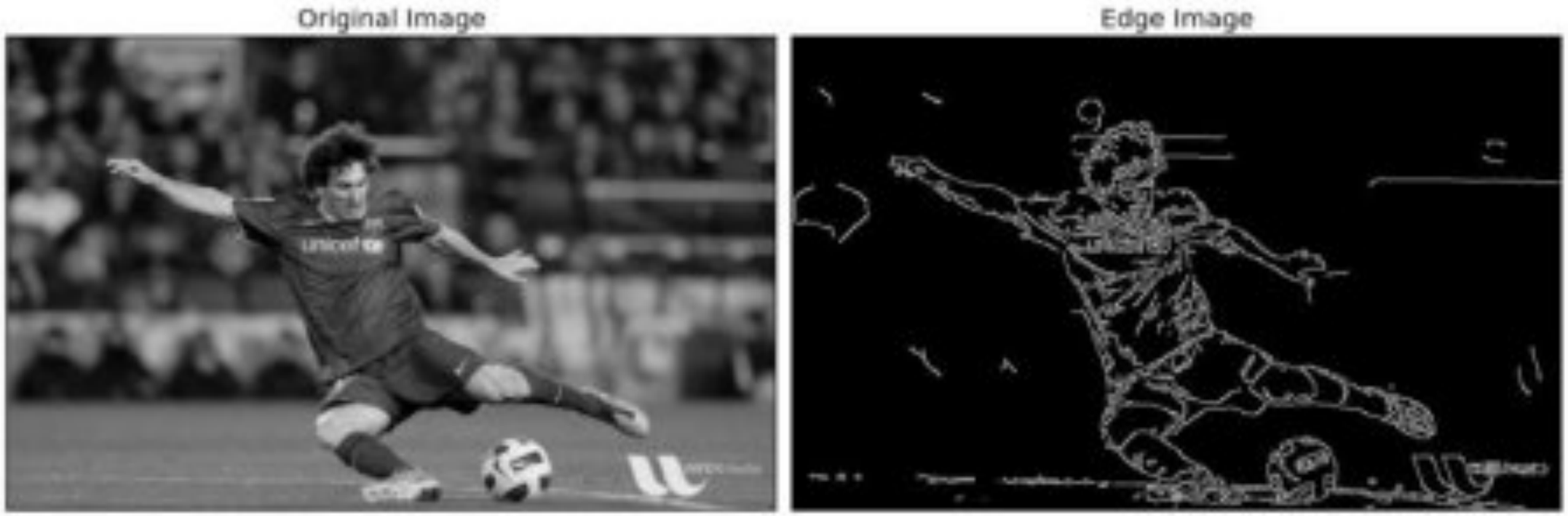
1.32638	1.77477	1.51587
2.83963	3.69403	3.07627
3.22121	4.14113	3.4107

Add these 9 values up, we will get the Gaussian Blur value of the center point.

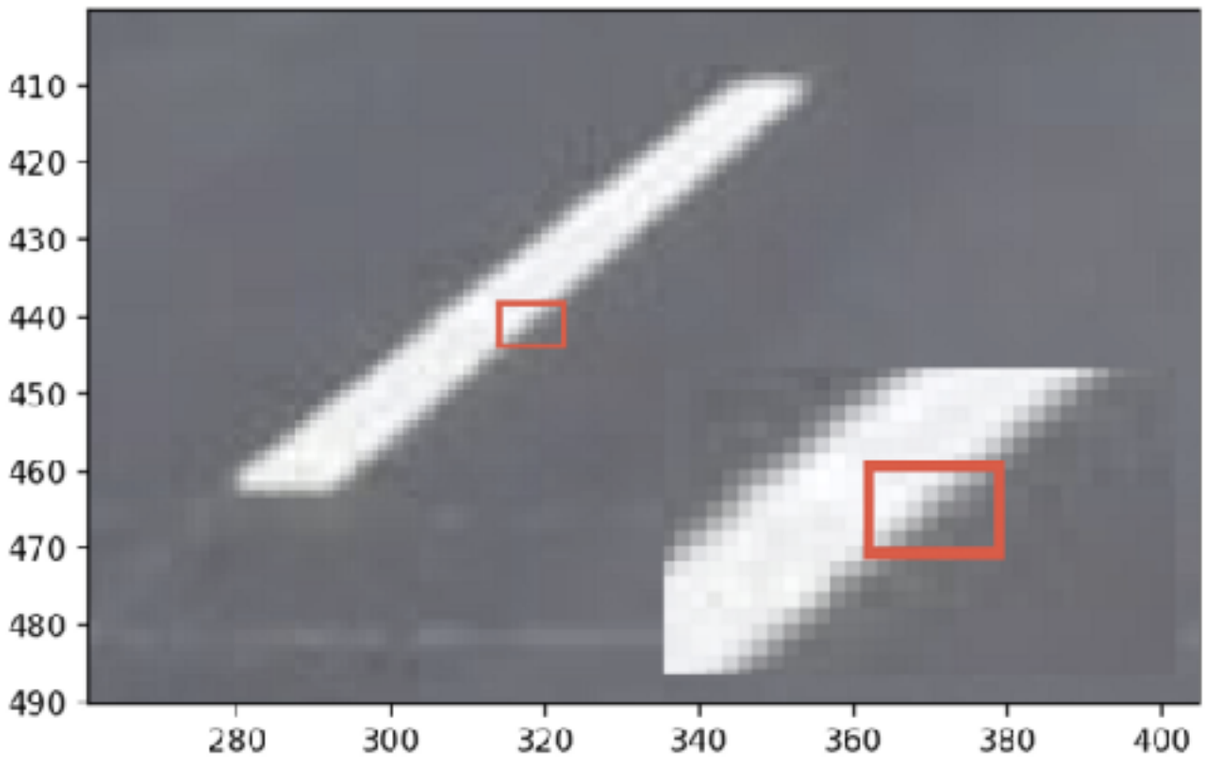
The parameters to a Gaussian blur are:

- **Sigma () – This defines how much blur there is. A larger number is a higher amount of blur.**
- **Radius – The size of the kernel in pixels**

Cany Edge Detection



The **Canny edge** detector is an **edge detection**operator that uses a multi-stage algorithm to **detect** a wide range of **edges** in images



If we pay close attention to the data surrounding this edge, it is fairly intuitive to conclude that edges are simply the *areas of an image where the color values change very quickly*. Therefore, detecting edges in an image becomes a mathematics problem of detecting any area where a pixel is a mismatch in color to all of its neighbors.

Canny also adds a pair of intensity threshold parameters which indicate generally how strong an edge must be to be detected.

Hough Transformation

The Hough Transformation is a great way to detect lines in an image. A thresholded edge image is the starting point for Hough transform.

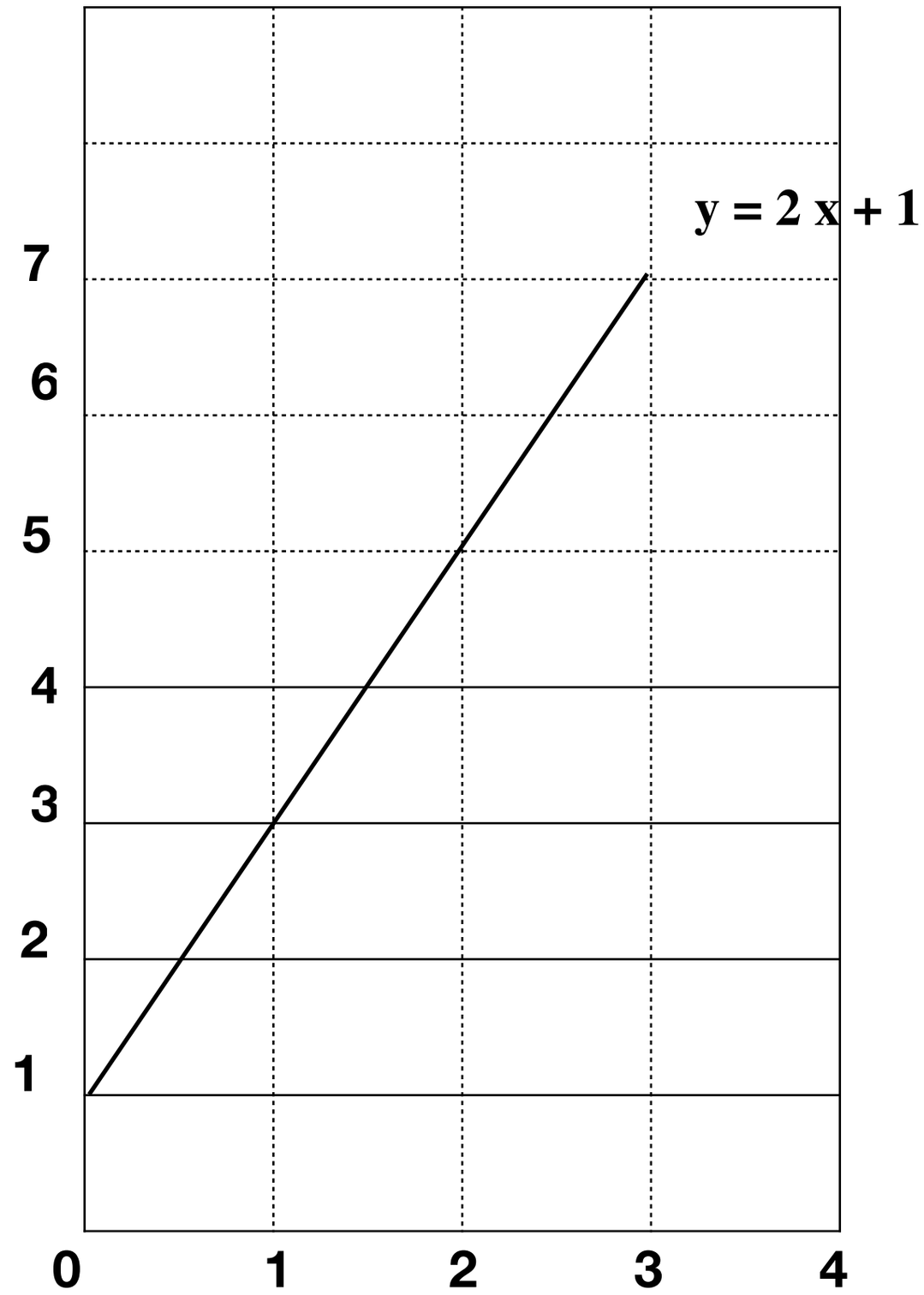
Cartesian Form

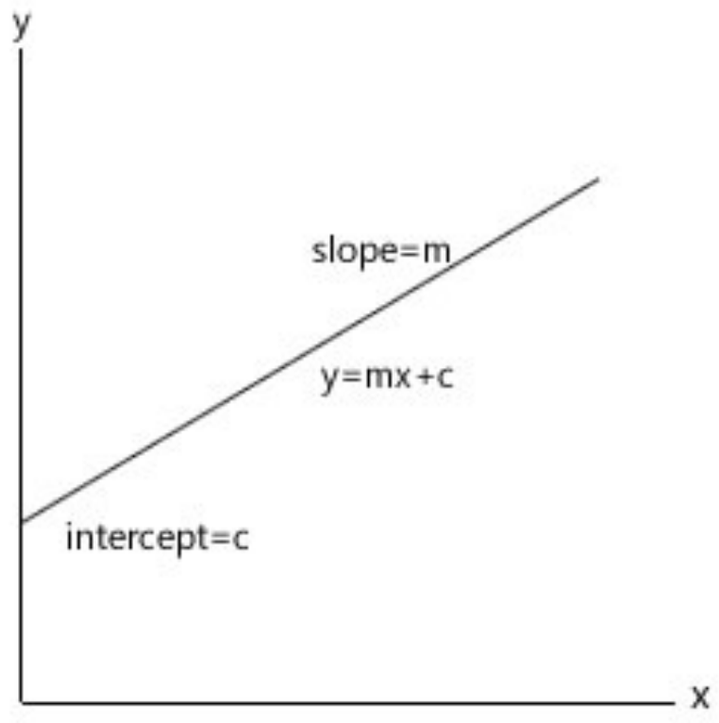
A line in Cartesian(also called rectangular) form has the equation:

y = mx + c

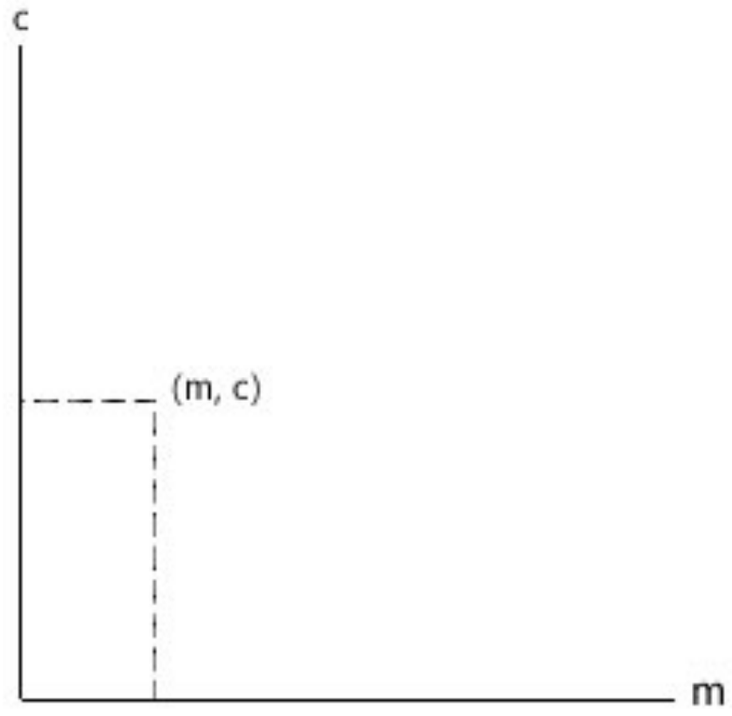
where:
c = y intercept
m = slope of the line

y	x
1	0
3	1
5	2
7	3





xy space



mc space

A line is a collection of points. And managing a collection of points is tougher than managing a single point. Using **m-c space (parameter space)** represent a line as a single point, without losing any information about it.

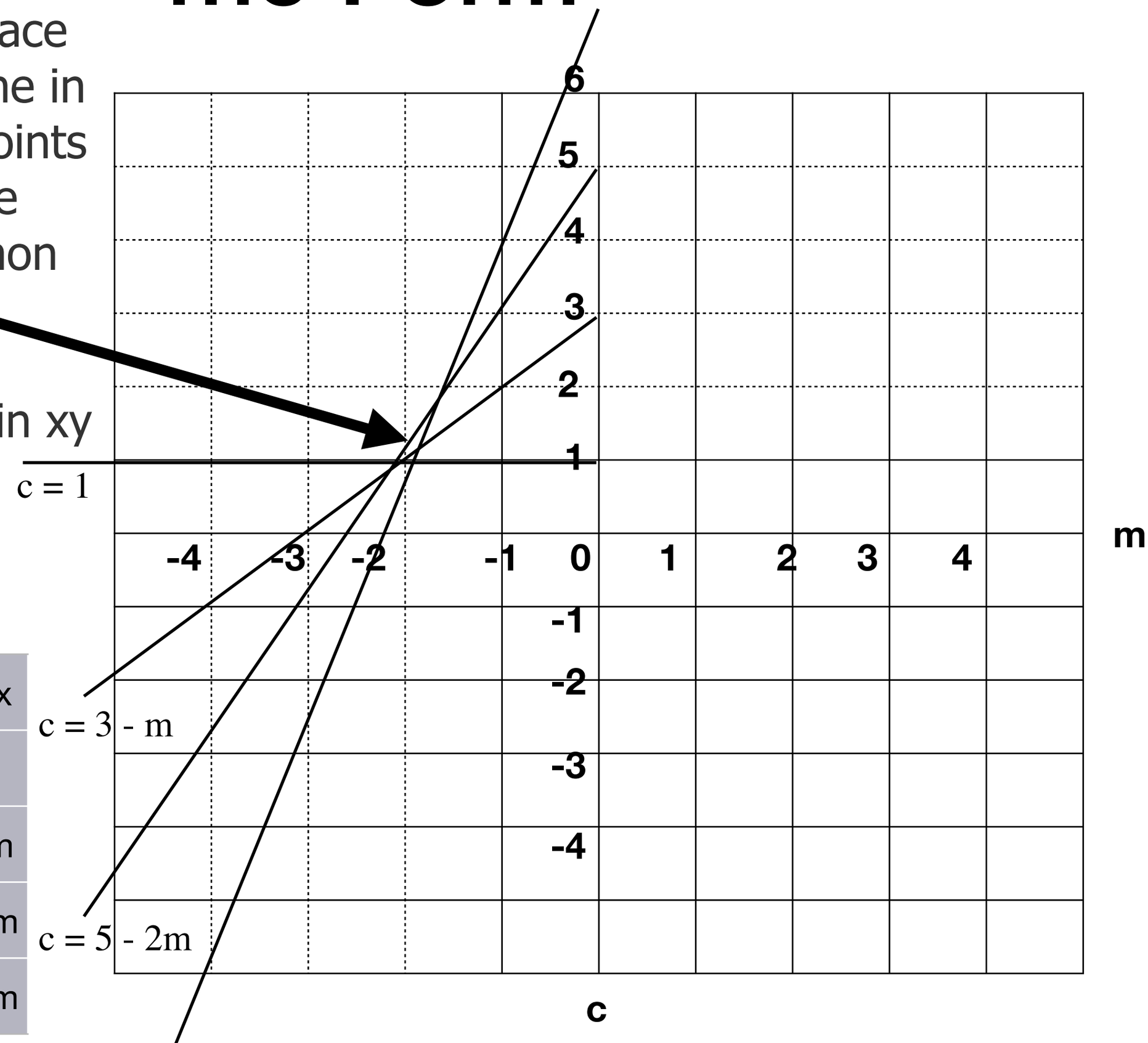
y = mx + c

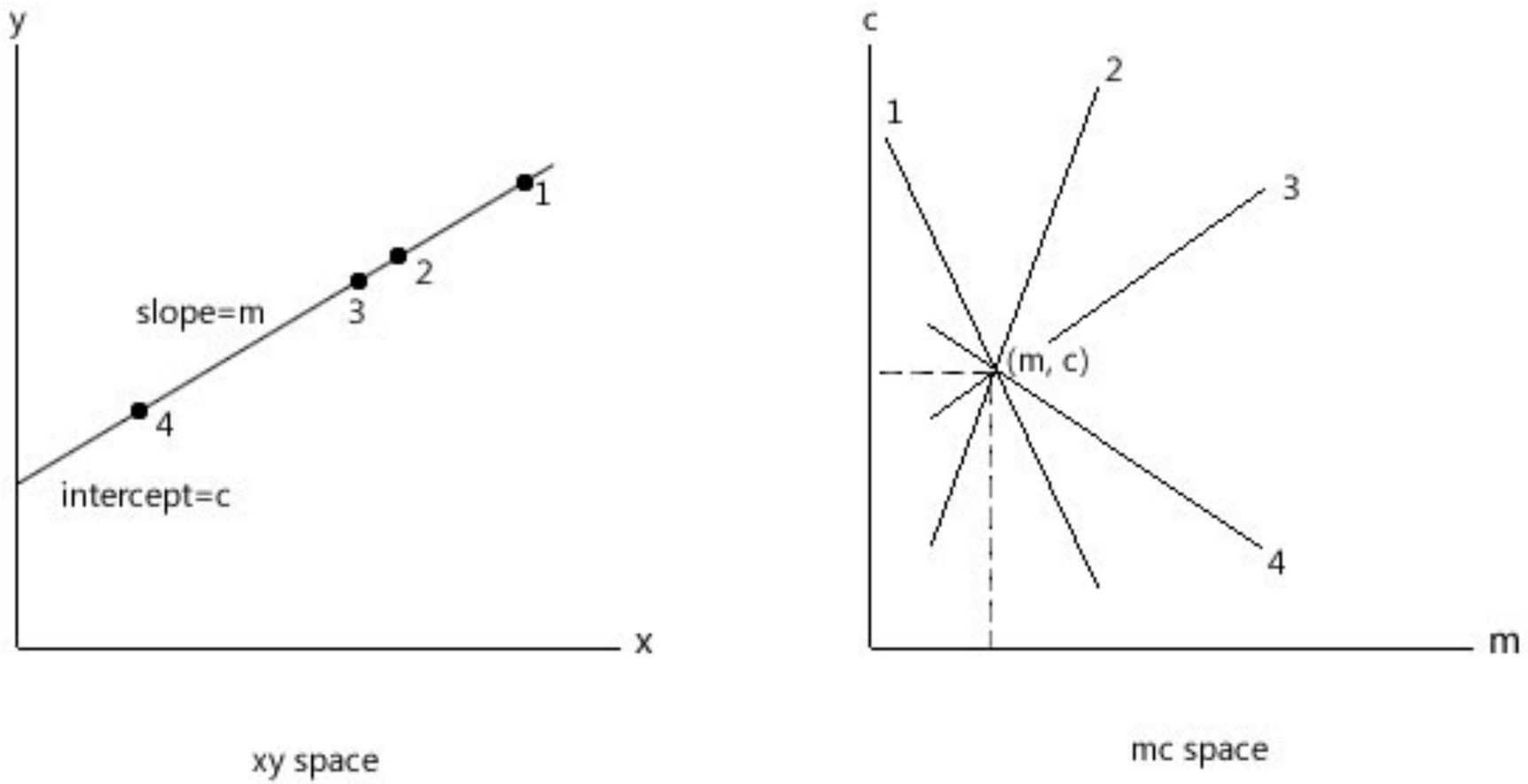
c = y - mx

mc Form

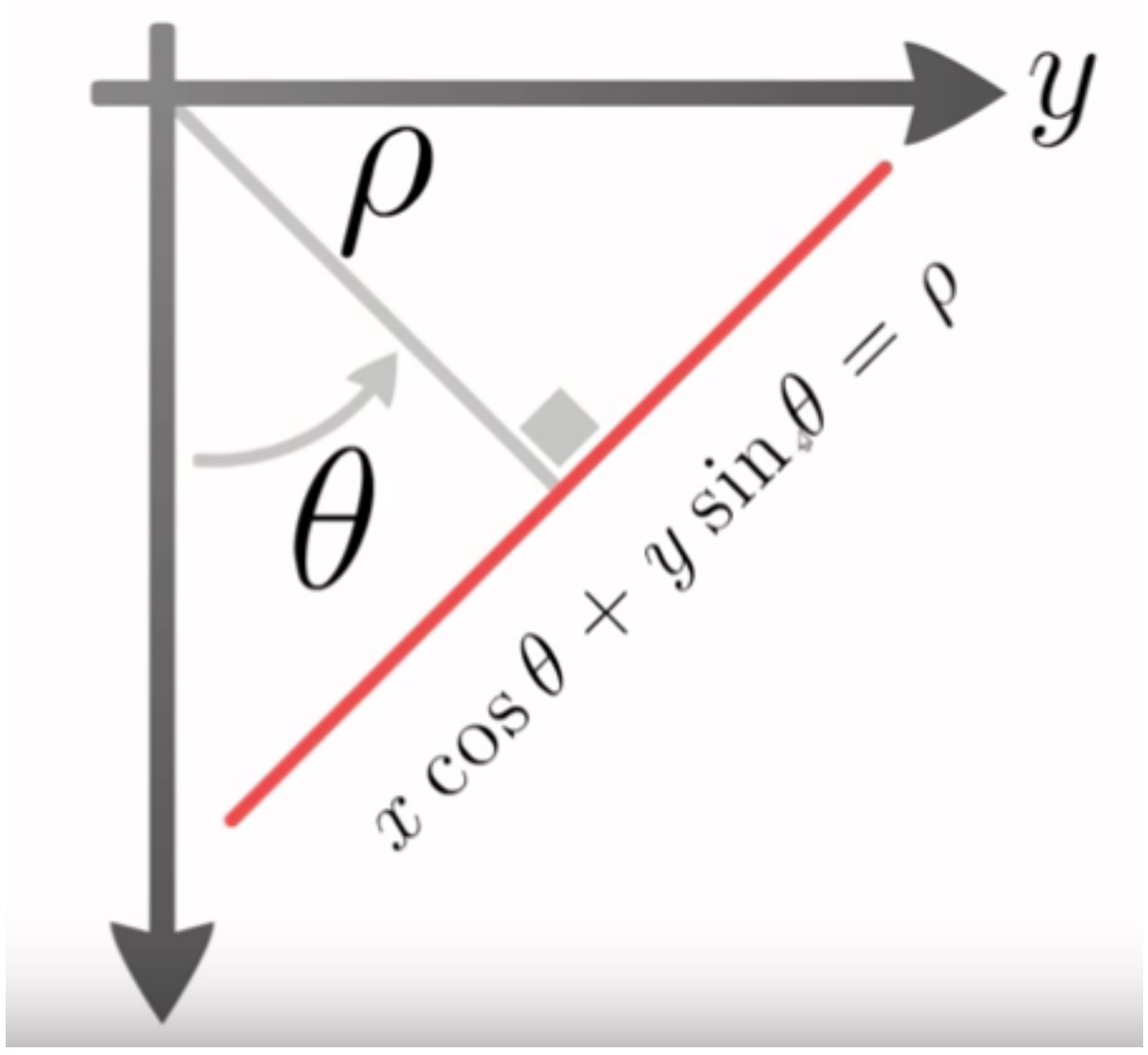
A point in the xy space is equivalent to a line in the mc space. All points on a line in xy space intersect at a common point in mc. This common point represents the line in xy space.

y	x	c = y-mx
1	0	c = 1
3	1	c = 3 - m
5	2	c =5 - 2m
7	3	c =7 - 3m



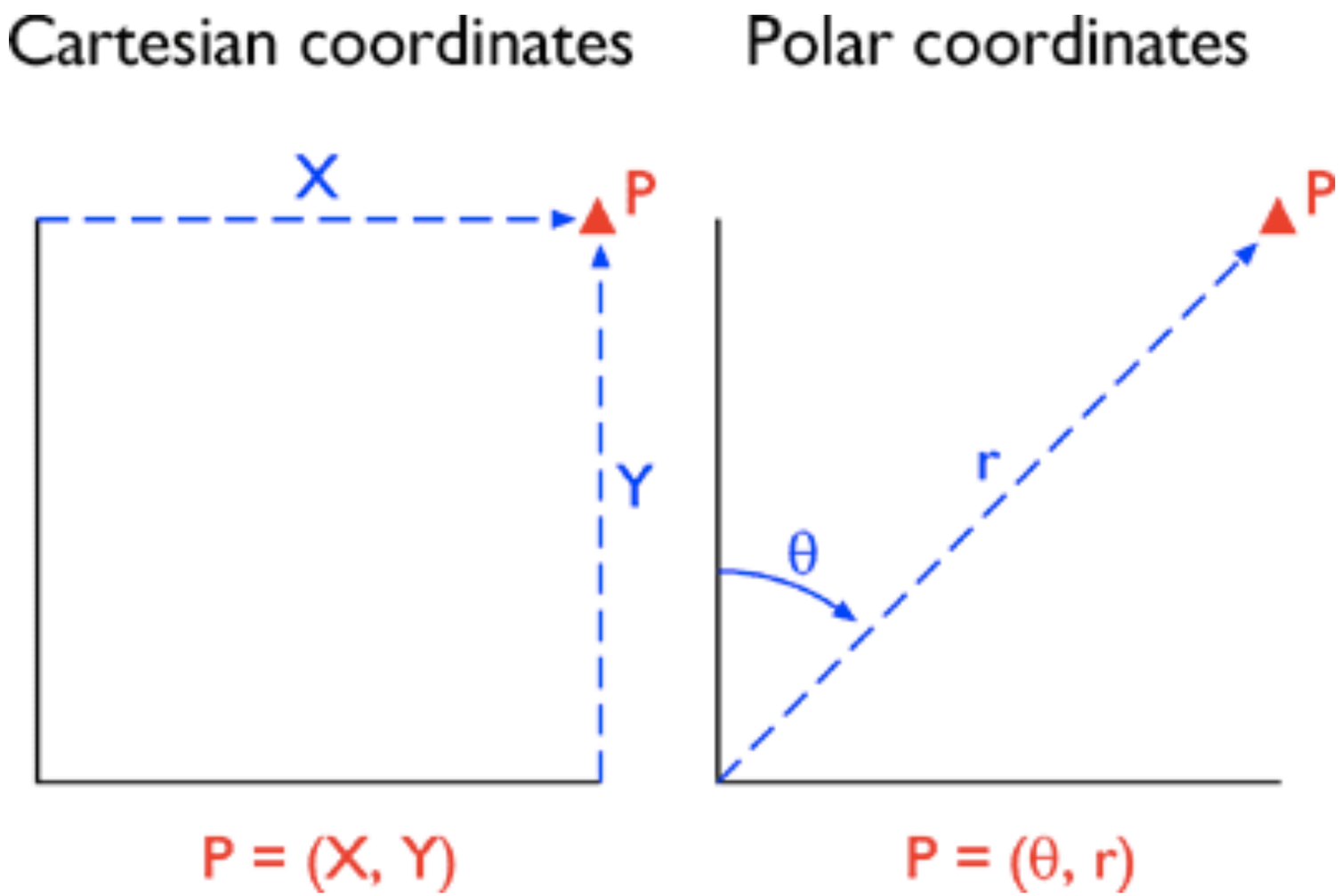


In a edge detected image, and for every point that is non black, when we draw lines in the mc space, some lines will intersect. These intersections mark are the parameters of the line.



Unfortunately, the slope is undefined when the line is vertical (division by 0!).To overcome this, we use another parameter space, the hough space (Polar Space).

Polar Form



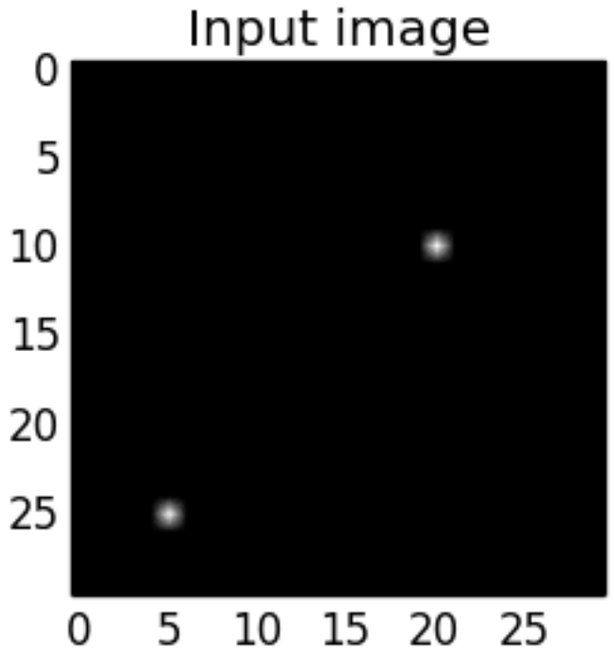
polar coordinates specifies the location of a point **P** in the plane by its distance **r** from the origin and the angle **theta** made between the line segment from the origin to P and the positive x axis. Every possible line can be represented by a unique r and θ. And further, every pixel on a given line will transform to the exact same r and θ for that line.

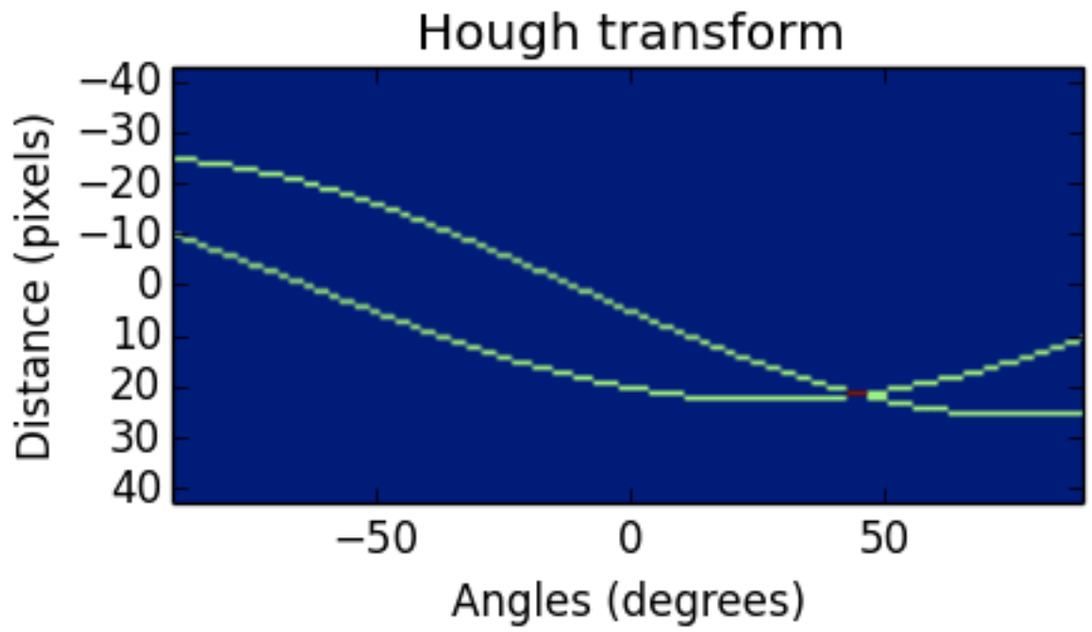
A line in the xy space is still equivalent to a point in the pθ space. But a point in the xy space is now equivalent to a sinusoidal curve in the pθ space.

Image space	Hough space
Straight line	Point
Point	Sinusoid

A point in the (x, y) space will thus be represented by a line in (r, θ)-space.

```
r = x*cos(theta) + y*sin(theta)
```



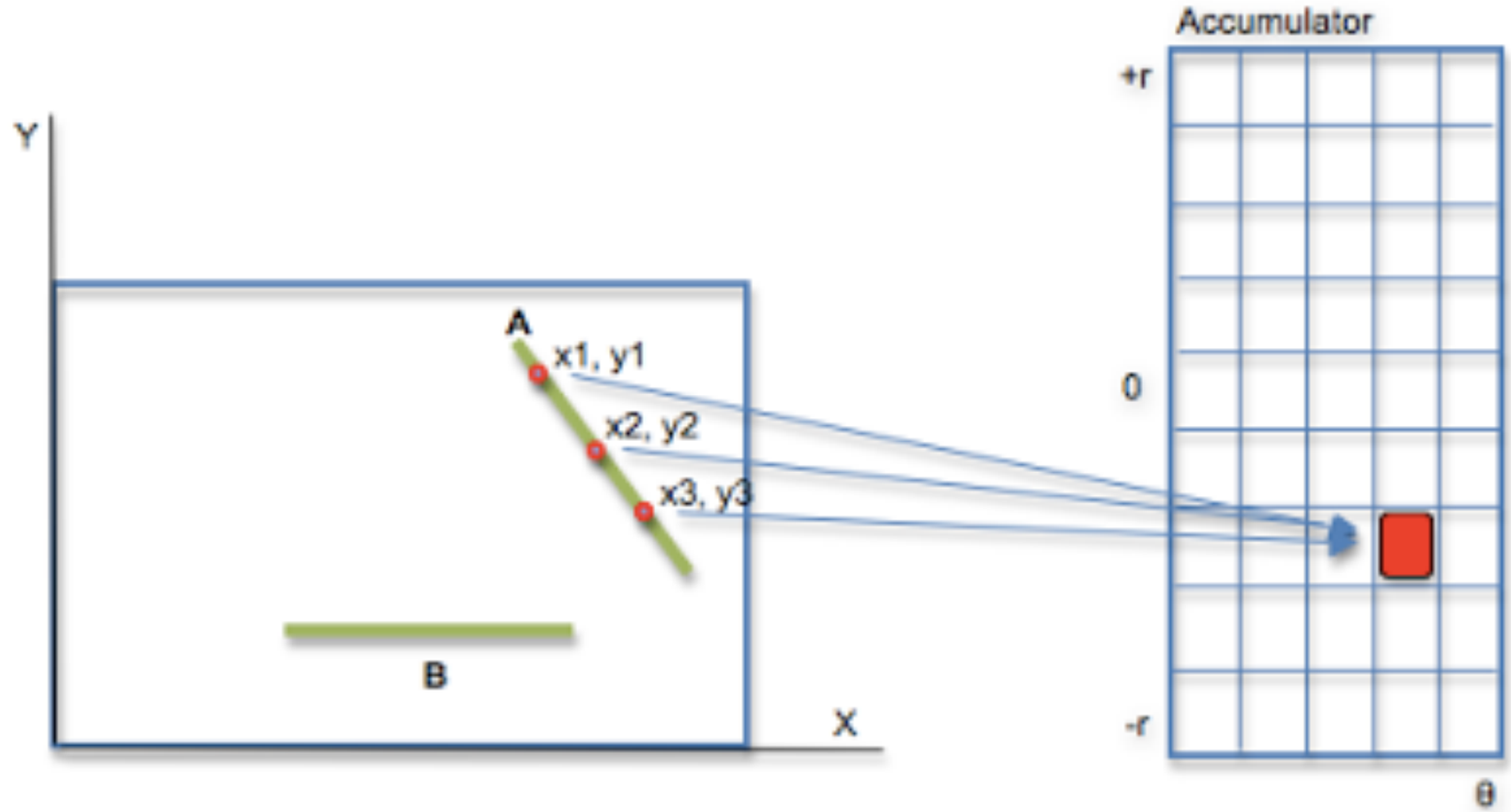


point	-90	-45	0	45	90
(5, 25)	-25	-25	5	21	25
(20, 10)	-10	7	20	21	10

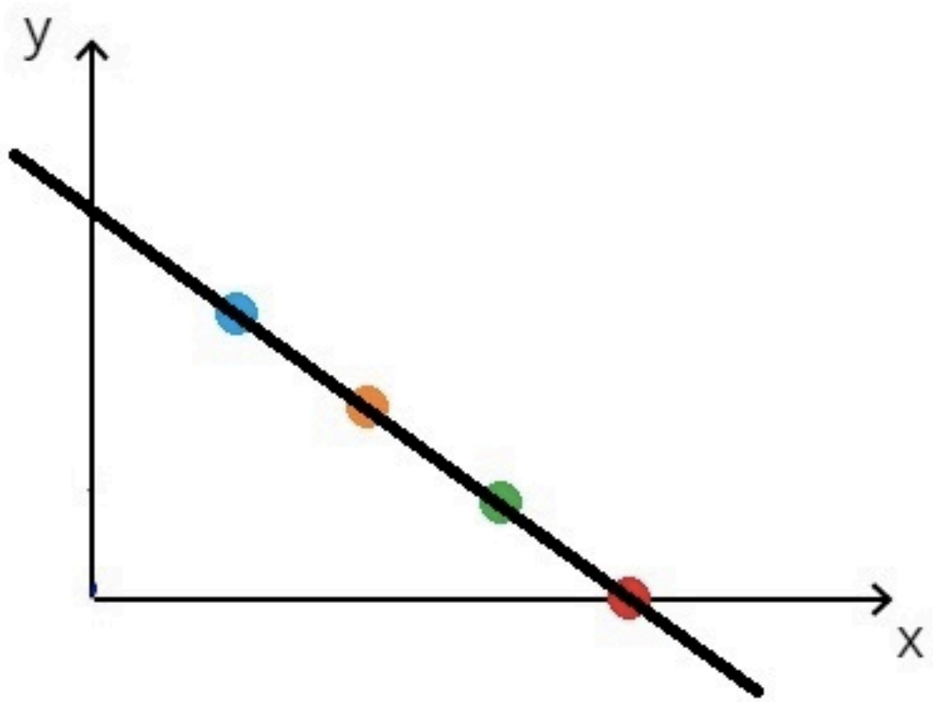
The points in hough space make a sinusoidal curve. The curves in hough space intersect at 45 degree with r=21. Curves generated by collinear points in the image space intersect in peaks(r,θ), in the Hough transform space. The more curves intersect at a point, the more “votes” a line in image space will receive.

Accumulator

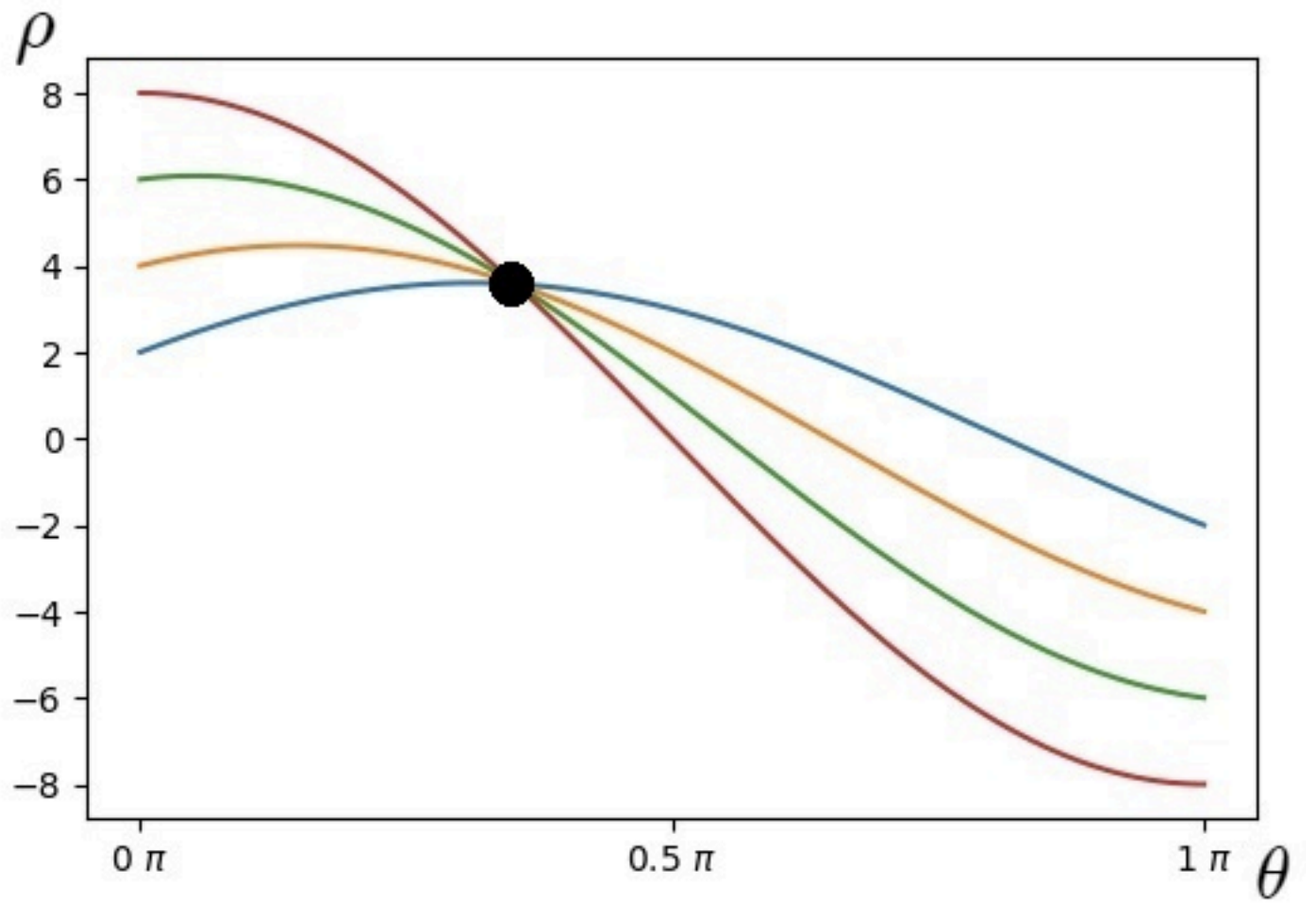
An Accumulator is nothing more than a 2 dimensional matrix with θmax columns and rmax rows. So each cell, or bin, represents a unique coordinate *(r, θ)* and thus one unique line in Hough Space.



- For every point in your image
  - compute r for θ = 1 to 180
  - increment the Accumulator by 1 for the every found r, so increment bin*(r, θ)*



Points which form a line



Bunch of sinusoids intersecting at one point

The result is a set of coordinates *(r, θ)*, one for every straight line in the image thats above the threshold. (for example, you can decide to only consider it a real line if it contains at least 80 pixels).

- you loop through every pixel of the edge detected image
- If a pixel is zero, you ignore it. It's not an edge, so it can't be a line. So move on to the next pixel.
- you take θ = -90 and calculate the corresponding p value. Then you "vote" in the accumulator cell (θ, p). That is, you increase the value of this cell by 1.
- Then you take the next θ value and calculate the next p value. And so on till θ = +90. And for every such calculation, making sure they “vote".
- So for every nonzero pixel, you'll get a sinusoidal curve in the pθ space
- The accuracy of the Hough transform depends on the number of accumulator cells you have. Say you have only -900, -450, 00, 450 and 900 as the cells for θ values. The "voting" process would be terribly inaccurate. Similarly for the p axis. The more cells you have along a particular axis, the more accurate the transform would be.