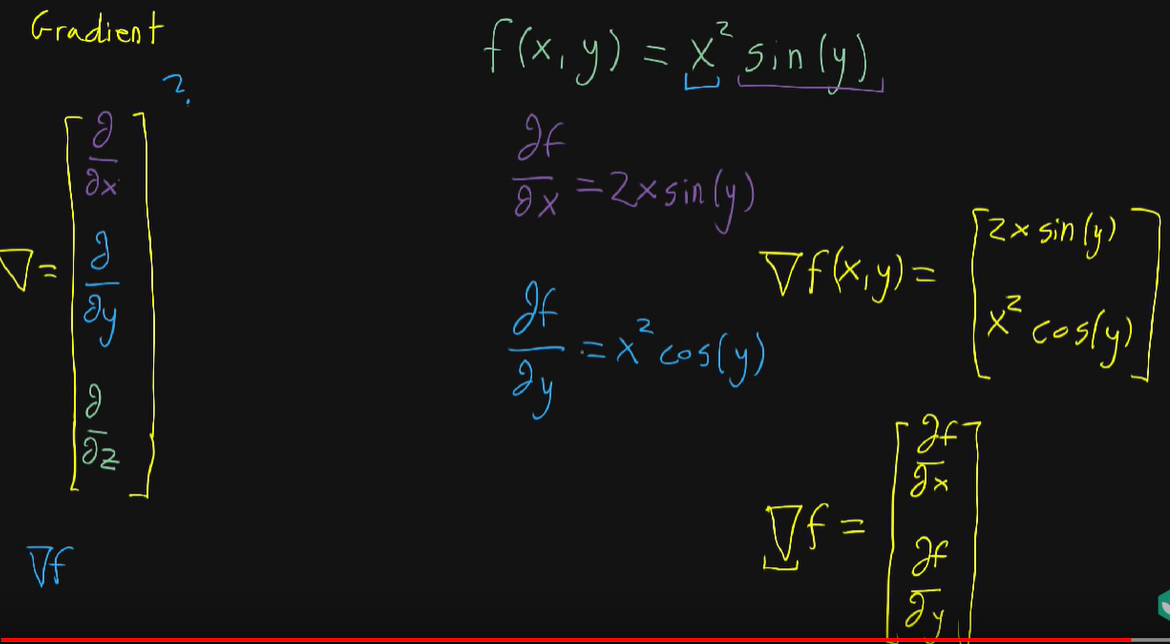
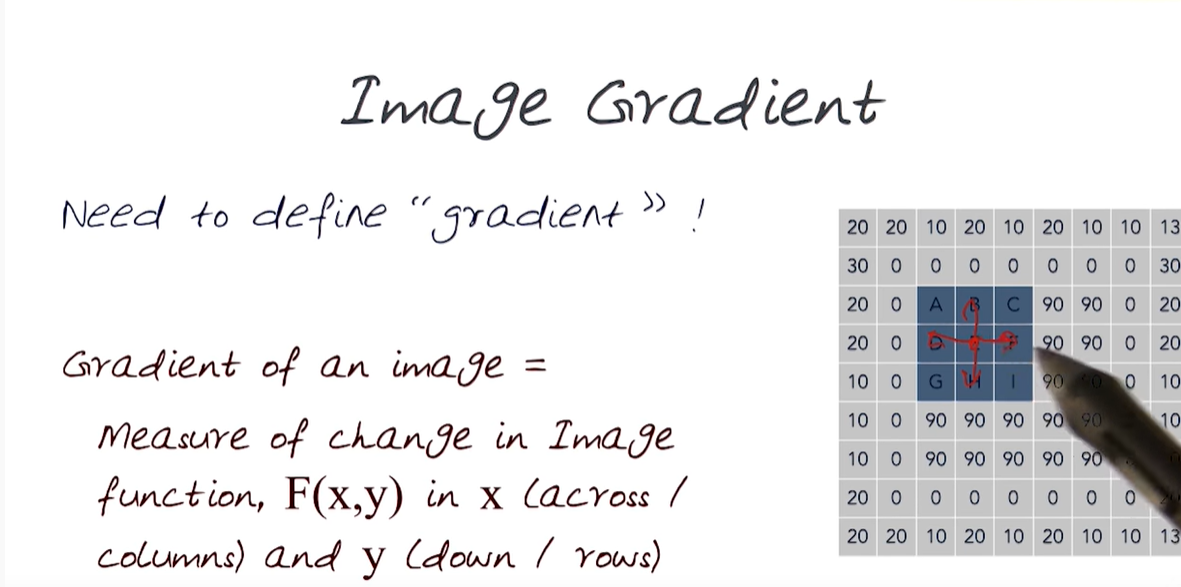
Gradients and Color Spaces

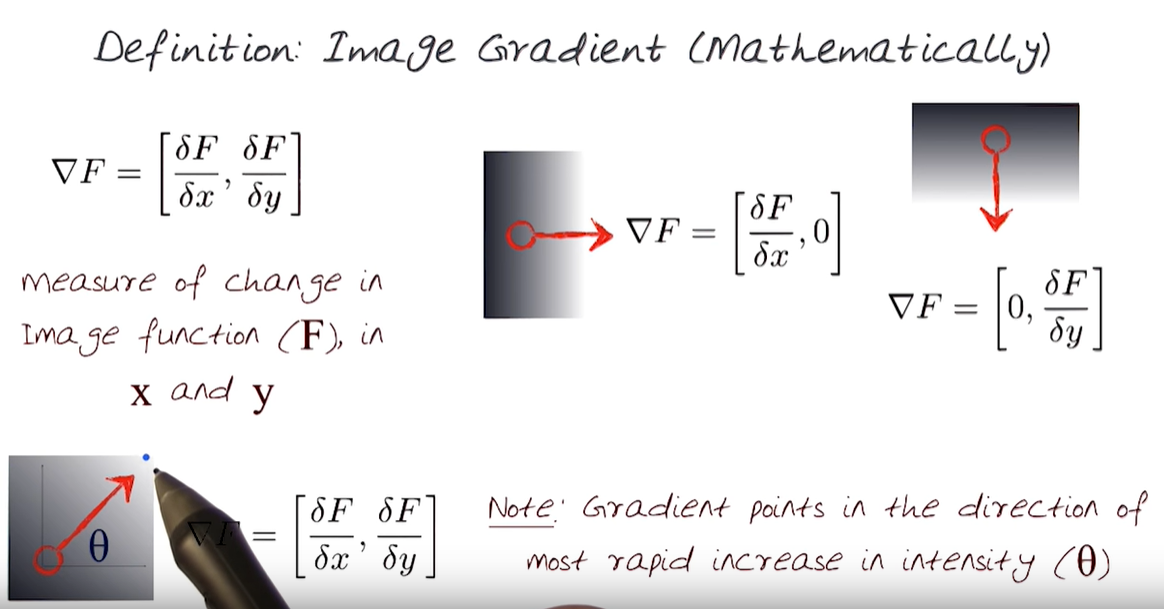
What is a gradient?: <https://www.youtube.com/watch?v=tIpKfDc295M>

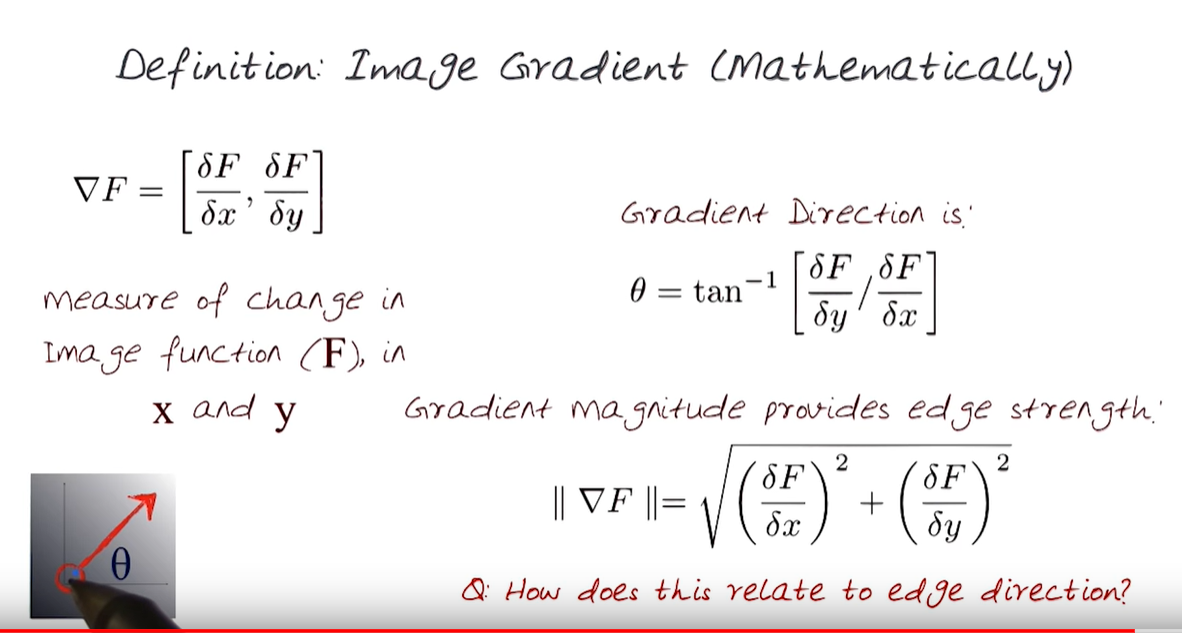


* Se ia o functie, iar gradientul sau este un vector format din derivatele partiale with respect(in functie) de x si y

Image Gradient: <https://www.youtube.com/watch?v=Dl5lPdoCXi8>







Other lectures (Edge Detection): <https://www.youtube.com/watch?v=7mEiTU-XgCo&t=1647s>

Filtering: <https://www.youtube.com/watch?v=1THuCOKNn6U&t=4s>

Other lectures of Computer Vision: <https://www.youtube.com/watch?v=715uLCHt4jE&list=PLd3hlSJsX_Imk_BPmB_H3AQjFKZS9XgZm>

Finding the Edges (Sobel Operator)

How Blurs & Filters Work: <https://www.youtube.com/watch?v=C_zFhWdM4ic> (aici o sa vezi ce este un **kernel convolution**)

**Kernel convolution** = is a process where we take a small grid of numbers and we pass them over the whole image. And by using different numbers in the kernel, we can perform blurs, or edge detection, or sharpen, unsharpen.

* Practic impulteste imaginea cu o alta matrice. Pentru fiecare pixel, impulteste cu matricea, insumeaza rezultatul si imparte la un kernel(ca filterul sa nu fie prea intunecos sau luminat), este un fel de a face o medie. Rezultatul este o valuare, ce va inlocui pixelul existent in imagine.
* Kernel size = se alege in general impar (este dimensionea matricei)

**Gaussian blur**: <https://www.youtube.com/watch?v=7LW_75E3A1Q>

**Gaussian filter**: <https://www.youtube.com/watch?v=-AuwMJAqjJc>

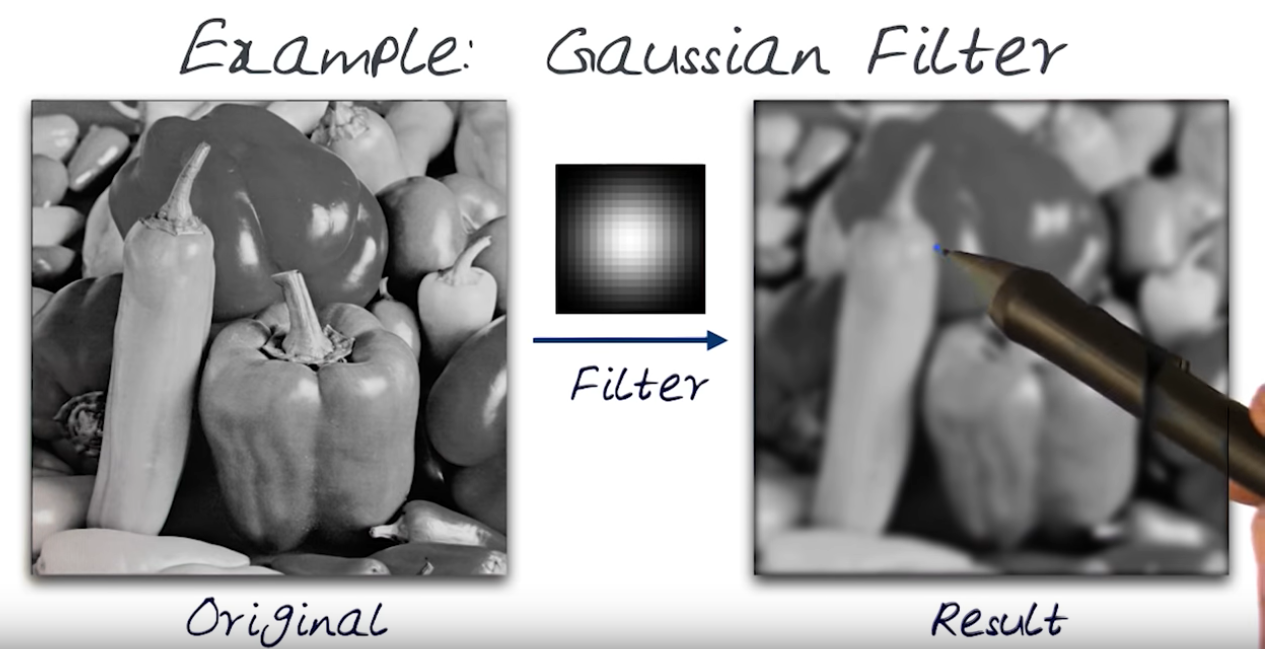
**Normal distribution** = alegi centrul, si pe masura ce te indepartezi, scade/creste intensitatea

Normal distribution of the Gaussian blur = este o curba

**Standard deviation** = the average distance from the mean of all the points. (determina cat de larga este curba)

Deci, cand aplici Gaussian filter, se blureaza imaginea, lasand ca marginile(Edges) sa fie mai itensificate.

**Gaussian filter**: <https://www.youtube.com/watch?v=-AuwMJAqjJc> (based on the equation of a Gaussian, it can be used to generate a kernel. Values: Gaussian or Normal Distribution)

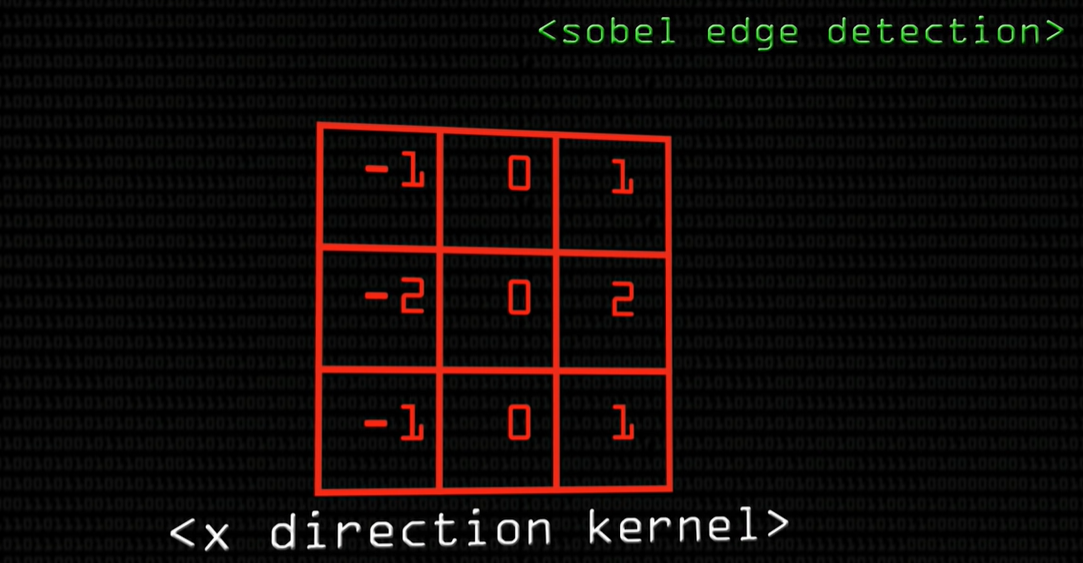


Finding the Edges(Sobel Operator): <https://www.youtube.com/watch?v=uihBwtPIBxM&t=3s>

**Edge Detection** = is a simply case of trying to find the regions in an image where we have a sharp change in intensity or a sharp change in color. A high value indicates a steep change and a low value indicates a shallow change.



**Sobel operator** (it is a very common operator) : is an approximation to a derivative of an image. So it separate the x, and y direction.

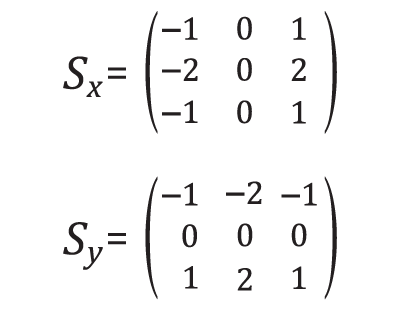


* The kernel has: minus operators on the left hand side, and positive numbers on the right hand side.
* Deci ce ne dorim este sa vedem schimbarea in imagine, in zonele de nord (-1,0,1) cu cele din E(-1,-2,-1) vezi video, de asta ai 0 in mijloc, ca sa te uiti in jurul pixelului.
* **It is not going to do anything to the edges**

In the video they say: Apply on Grayscale, Apply Gaussian blur first, before sobel edge detector just to get rid of the low and high frequency stuff and keep the low frequency big Walls.

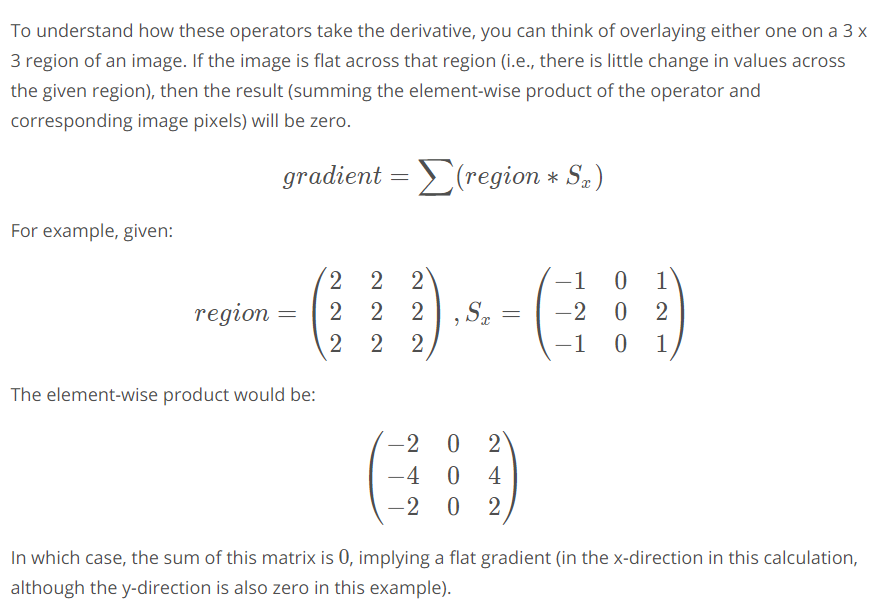


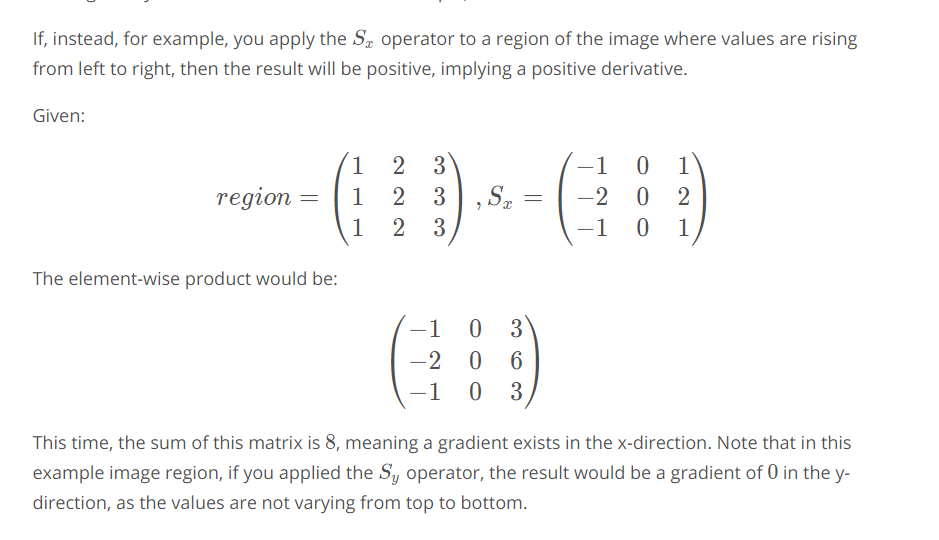
Din curs, Sobel Algorithm:

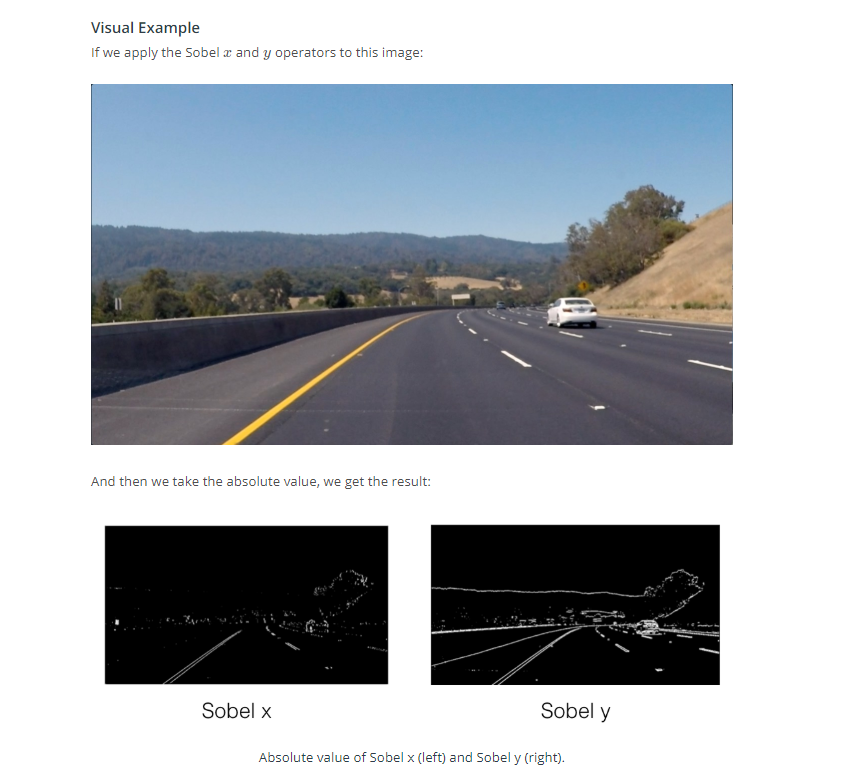
* The Sobel operator is at the heart of the Canny edge detection algorithm you used in the Introductory Lesson. Applying the Sobel operator to an image is a way of taking the derivative of the image in the x*x* or y*y* direction. The operators for Sobel\_x*Sobelx*​ and Sobel\_y*Sobely*​, respectively, look like this:
* [[](https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/5d1efbaa-27d0-4ad5-a67a-48729ccebd9c/lessons/144d538f-335d-454d-beb2-b1736ec204cb/concepts/e6115672-155d-4c10-b640-fe20a4f4b0a6)](https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/5d1efbaa-27d0-4ad5-a67a-48729ccebd9c/lessons/144d538f-335d-454d-beb2-b1736ec204cb/concepts/e6115672-155d-4c10-b640-fe20a4f4b0a6)

This is the minimum size, but the kernel size can be any odd number.

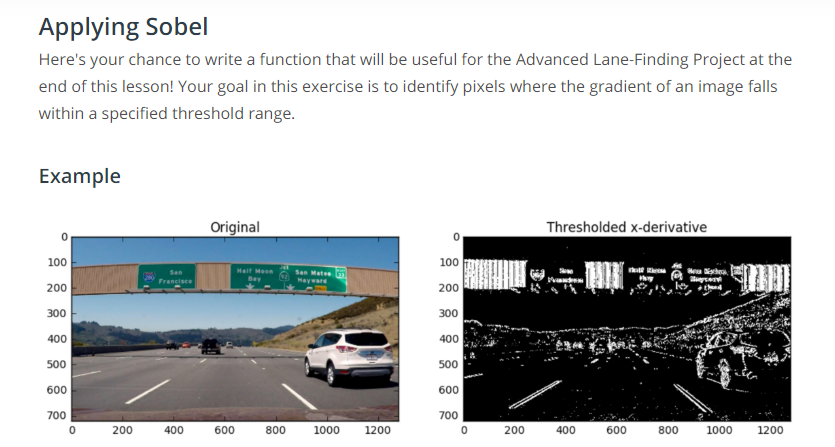
A larger kernel implies taking the gradient over a larger region of the image, or, in other words, a smoother gradient.







Quiz:



Pasi:

**def** **abs\_sobel\_thresh**(img, orient='x', thresh\_min=0, thresh\_max=255):

*# Grayscale*

*# Apply cv2.Sobel()*

*# Take the absolute value of the output from cv2.Sobel()*

*# Scale the result to an 8-bit range (0-255)*

*# Apply lower and upper thresholds*

*# Create binary\_output*

**return** binary\_output

ass in img and set the parameter orient as 'x' or 'y' to take either the x*x* or y*y* gradient. Set thresh\_min, and thresh\_max to specify the range to select for binary output. You can use exclusive (<, >) or inclusive (<=, >=) thresholding.

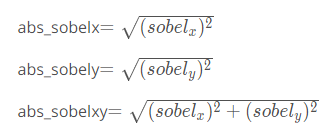
**NOTE:** Your output should be an array of the same size as the input image. The output array elements should be 1 where gradients were in the threshold range, and 0 everywhere else



**Magnitude of the Gradient**

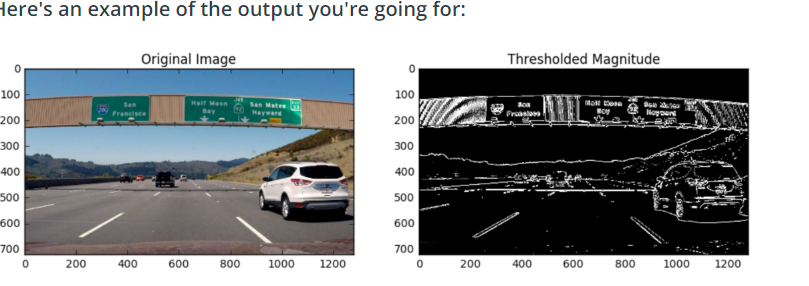
* In quizul trecut am vazut cum putem sa luam gradientul in x sau y si sa stabilim thresholds pentru a identifica pixelii intr-o anumit interval in functie de gradient.
* Dacac ne jucam cu gradientul in functie de x, o sa observam ca realizeazam un “cleaner job” pentru a identifica liniile.

**Magnitude of the gradient** =  or absolute value, of the gradient is just the square root of the squares of the individual x and y gradients. For a gradient in both the x*x* **and** y*y* directions, the magnitude is the square root of the sum of the squares..



! Taking the gradient over larger regions can smooth over noisy intensity fluctuations on small scales. The default Sobel kernel size is 3, but here you'll define a new function that takes kernel size as a parameter.

! It's important to note here that the kernel size should be an **odd** number. Since we are searching for the gradient around a given pixel, we want to have an equal number of pixels in each direction of the region from this central pixel, leading to an odd-numbered filter size - a filter of size three has the central pixel with one additional pixel in each direction, while a filter of size five has an additional two pixels outward from the central pixel in each direction.

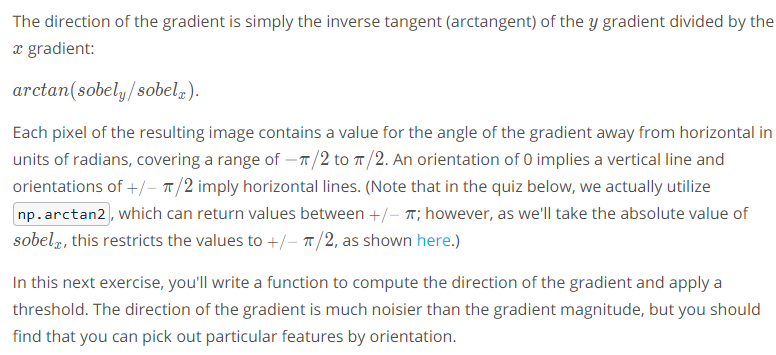




**Direction of the Gradient**

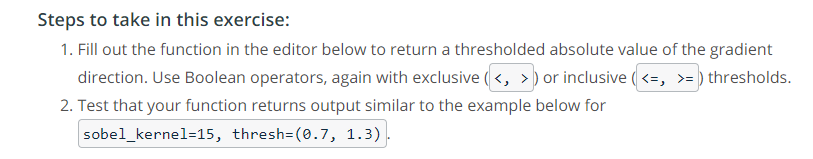
* Anterior am vazut ca in momentul in care aplicam gradientul si magnitudinea, obtinem liniile din imagine foarte bine, dar detecteaza si alte lucruri
* Gradient magnitude sta la baza Canny edge detection

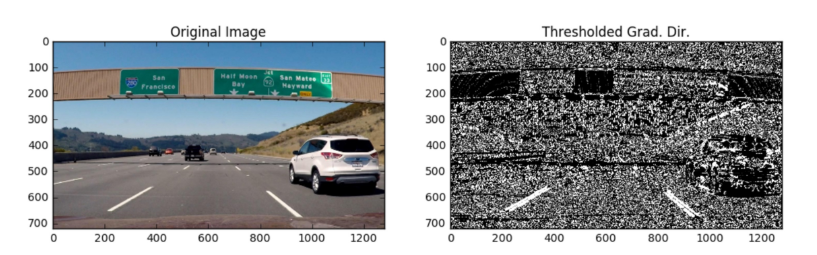
In the case of lane lines, we're interested only in edges of a particular orientation. So now we will explore the direction, or orientation, of the gradient.



Atan2 : <https://en.wikipedia.org/wiki/Atan2>









**Combining Thresholds**

If you play around with the thresholds in the last exercise, you'll find that you can start to identify the lane lines by gradient direction alone by setting the threshold around thresh = (0.7, 1.3), but there's still a lot of noise in the resulting image.

Now consider how you can use various aspects of your gradient measurements (x, y, magnitude, direction) to isolate lane-line pixels. Specifically, think about how you can use thresholds of the x and y gradients, the overall gradient magnitude, and the gradient direction to focus on pixels that are likely to be part of the lane lines.

**(aici ai un mic proiectel**)

**Color Spaces**

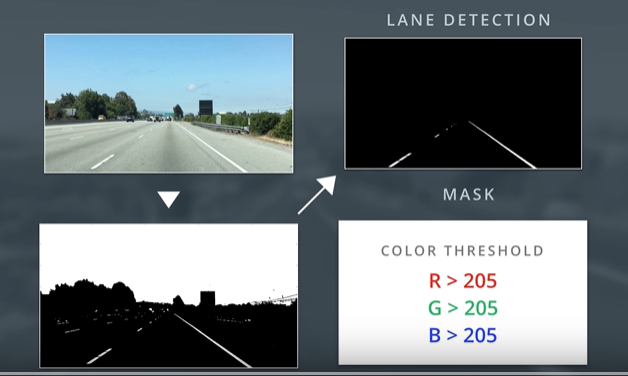
* Anterior noi converteam in GRAYSCALE imaginea, si pierdeam culorile la linii (de exemplu galben), acum vrem sa revenim si sa indentificam si culorile



**Color Thresholding**

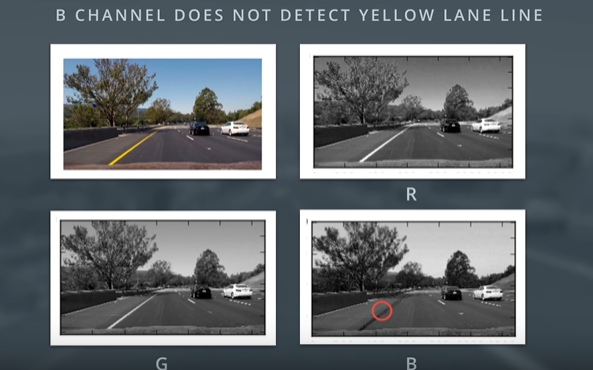
**Video:** <https://www.youtube.com/watch?v=dMI_so4P1Jc>

**RGB** is red-green-blue color space. You can think of this as a 3D space, in this case a cube, where any color can be represented by a 3D coordinate of R, G, and B values. For example, white has the coordinate (255, 255, 255), which has the maximum value for red, green, and blue.



* Anterior noi am folosit diferite masti si color thresholds pe aceste valori RBG pentru a identifica pixeli cei mai luminosi ce alcatuiesc o linie in imagine (“bright white lane pixels”)
* This lane detection can work well alongside gradient detection which relies on grayscale intensity measurements.
* Dar, RGB nu functioneaza bine cand avem light conditions, sau cand liniile au culori diferite, de exemplu: galben.

**De exemplu:**

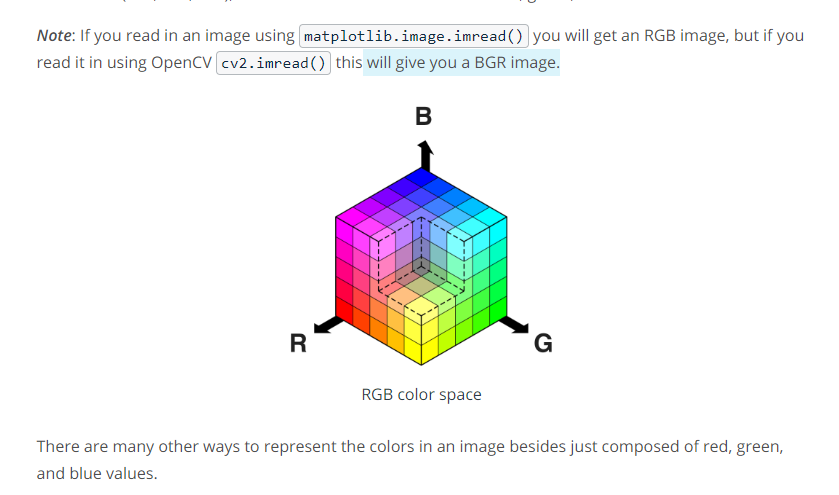


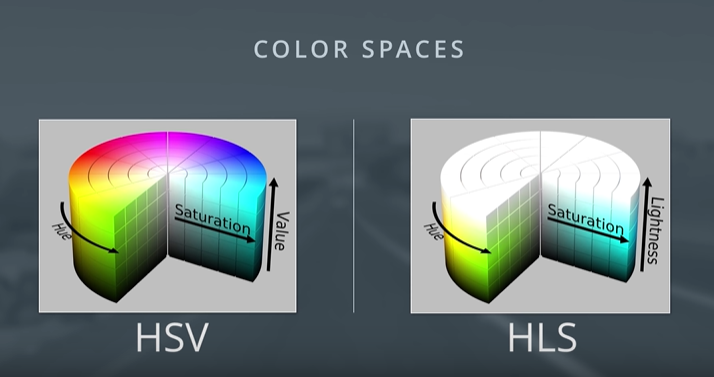
* Daca luam imaginea color si o descompunem in RGB, observam ca linia galbena apare doar in R si G, iar la B nu e nimic.



* Dar observam ca pt fiecare imagine, daca ne uitam in departare, au diferite nivele de brightness, ceea ce ne face imposibil sa ne dam seama ce culori sunt in spate.

A **color space** is a specific organization of colors; color spaces provide a way to categorize colors and represent them in digital images.





* **HSV = (hue, staturation, value)**
* **HLS = (hue, lightness, saturation)**

**Hue = the dominant color as perceived by an observer**

**Saturation = the amount of white light mixed with a Hue (birght of a color), how pure a color is**

**Value = the chromatic notion of intensity**

**Lightness = efectiv cum ai pune un bec asupra culorilor, mai intunecat, sau nu**

**Video(HSV):** <https://www.youtube.com/watch?v=QA2dLTOJfJk>

**Video(HLS):** <https://www.youtube.com/watch?v=Ceur-ARJ4Wc>

These are some of the most commonly used color spaces in image analysis.

To get some intuition about these color spaces, you can generally think of **Hue** as the value **that represents color independent of any change in brightness**. So if you imagine a basic red paint color, then add some white to it or some black to make that color lighter or darker -- the underlying color remains the same and the hue for all of these colors will be the same.

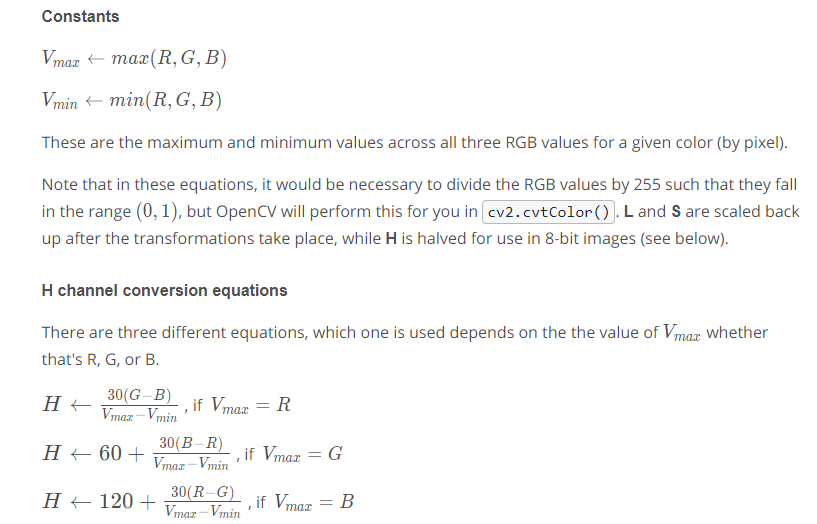
On the other hand, **Lightness** and **Value** represent different ways to measure the relative lightness or darkness of a color. For example, a dark red will have a similar hue but much lower value for lightness than a light red.

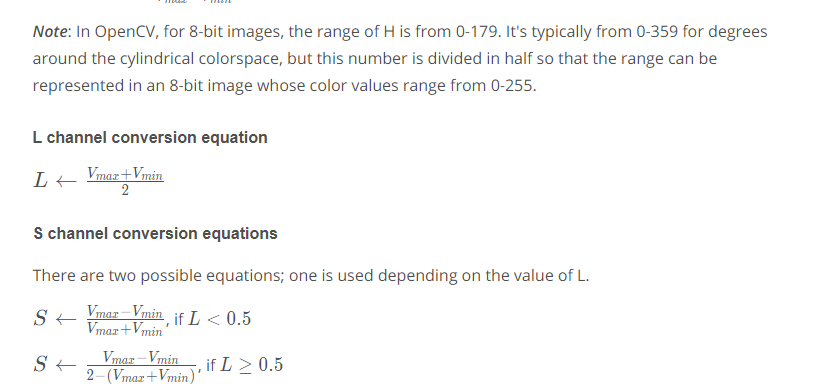
**Saturation** also plays a part in this; saturation is a measurement of colorfulness. So, as colors get lighter and closer to white, they have a lower saturation value, whereas colors that are the most intense, like a bright primary color (imagine a bright red, blue, or yellow), have a high saturation value. You can get a better idea of these values by looking at the 3D color spaces pictured below. (la mine mai sus)

Most of these different color spaces were either inspired by the human vision system, and/or developed for efficient use in television screen displays and computer graphics.

In the code example, **I used HLS space to help detect lane lines of different colors and under different lighting conditions.**

OpenCV provides a function hls = cv2.cvtColor(im, cv2.COLOR\_RGB2HLS) that converts images from one color space to another. If you’re interested in the math behind this conversion, take a look at the equations below; note that all this math is for converting 8-bit images, which is the format for most road images in this course. These equations convert one color at a time from RGB to HLS.



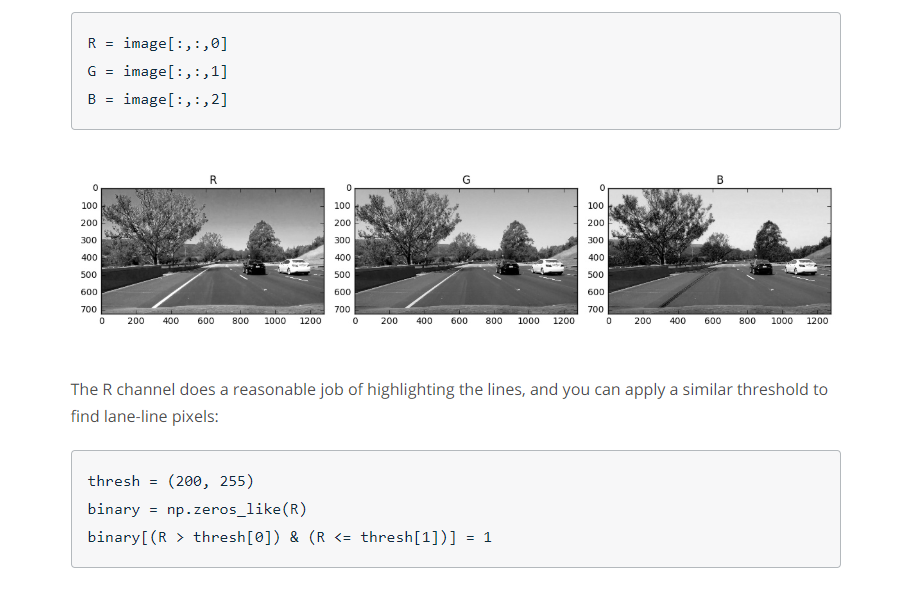


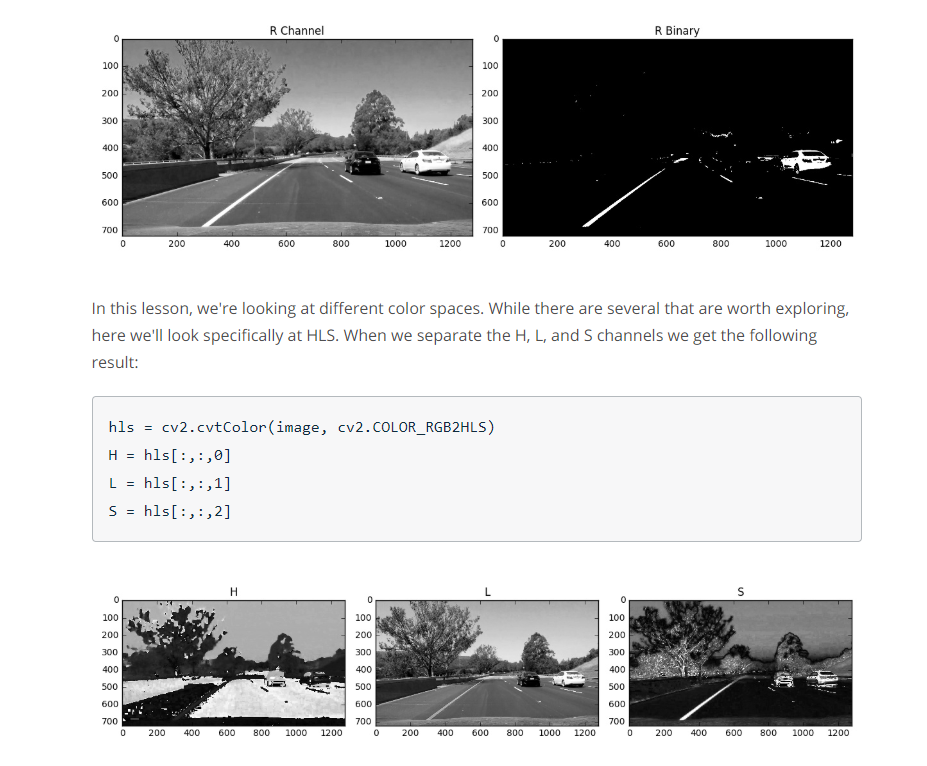
**HLS and Color Thresholds**

You've now seen that various color thresholds can be applied to find the lane lines in images. Here we'll explore this a bit further and look at a couple examples to see why a color space like HLS can be more robust. Let's first take another look at some of the images you saw in the last video.

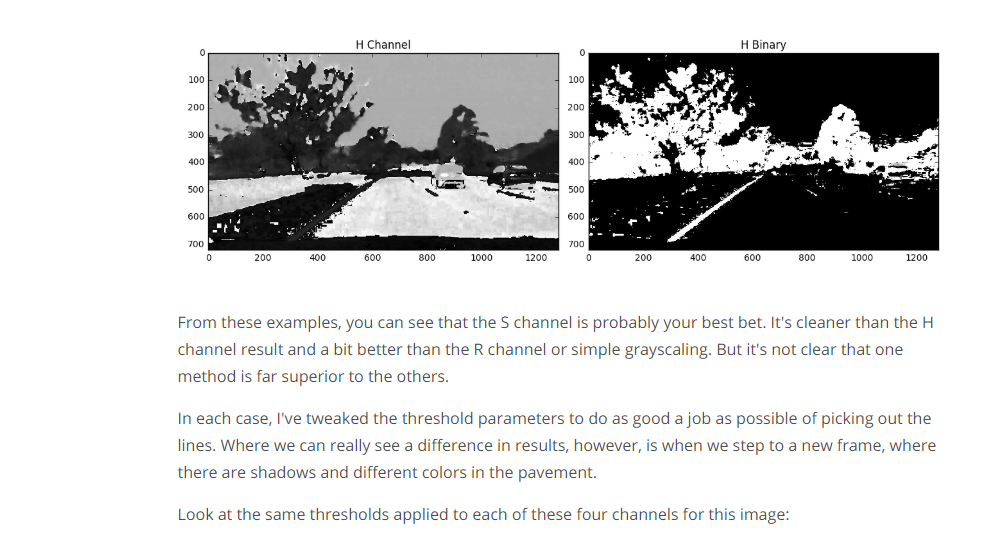
Here I'll read in the same original image (the image above), convert to grayscale, and apply a threshold that identifies the lines:















****

**Color And Gradient**

