3 Steps of Robotics

1. Sense/ Percieve the world/data (Computer Vision is very important for perception)
2. Decide what to do based on what is percieved
3. Perform an action based on the Desisicon

Self-Driving car: lane markings, pedestrians, other vehicles, and more need to be percieved

Why bother doing perception with only a camera when we can also do it with radar and LIDAR data?

That is due to the cost of cameras when compared to those technologies.

Also Radar and LIDAR see the world in 3D while a Camera can only see in 2D. But the Spatial Resolution of a Camera is much higher.

In fact that Spatial Resolution is so high that we can actually infer 3D Data from a camera.

**Overview:**

1. Upgrade our Lane Lines Project so that it can deal with much more complex scenarios. Like Curving lines, shadows, change in pavement color, how much vehicle is curving, and how far it is from center
2. Implement vehicle detection and tracking, so that we can make decisions like when to change lanes.
3. Combine both to simultaneously measure where the car is on the road, where the road is going, and the location of other vehicles in our field of view.

Finding Lane Lines is the first step in measuring some of the quantities that need to be known in order to control the car.

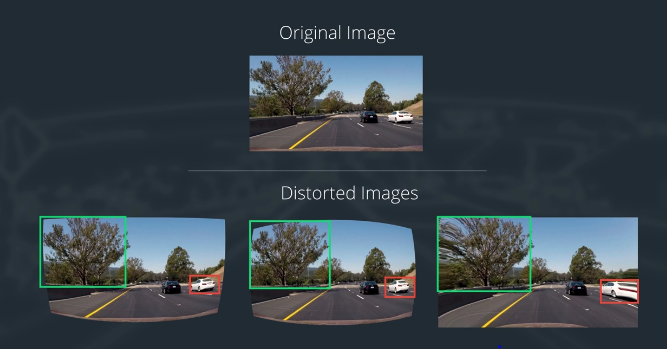
Ex. Steering a car requires info on how much our lane is curving. To do this we need to map out the lanes after transforming the perspective of the image received by the camera. Like for example transforming the perspective of the image to a top down view of the car. In order to get this perspective correct, we need to correct for the effect of Image Distortion.

Cameras don’t create perfect Images, in fact some of the objects in those images, especially near the edges can get stretched or skewed in various ways. These can be corrected with Distortion Correction.

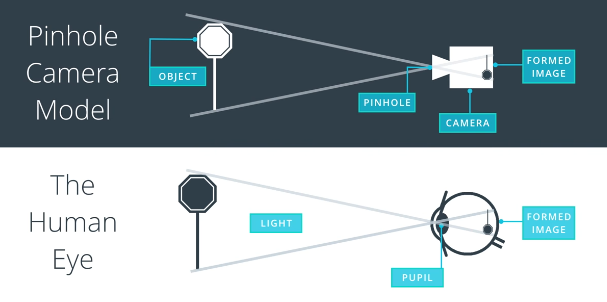
**Distortion:**

Image Distortion occurs when a camera looks at 3D objects in the real world and transforms them into a 2D image. The Distortion actually changes the shape and size of these 3D objects.

We need to correct these Distortions in order to get correct and useful information out of our cameras.

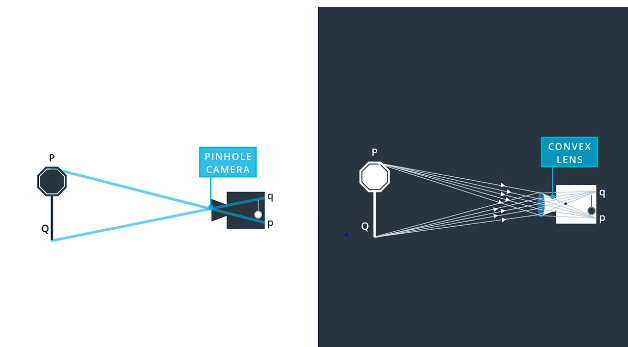


**Pinhole Camera Model:**



Camera model that emulates the human eye is called the pinhole camera model. Through the small pinhole in the camera, the camera focuses the light from a 3D object to a 2D Image at the back of the camera and is read using Film or a Sensor. This Image is also upside down and reversed due to the light rays from the top of the object being at the bottom of the sensor.

The transformation of a 3D object point P(x,y,z) to a 2D image p(x,y) is done by a transform matrix called the camera matrix. This Camera matrix is required to calibrate the camera later on.

Real cameras don’t use pinholes to focus light onto the sensor, but they use lenses. Lenses focus multiple light rays onto the sensor at the same time, but the bending of a light ray in the edges of the lens cause distortion.

The Pinhole model of a camera is free from distortions while the real life use of lenses introduces distortions.

**Types of Distortion:**

**Radial Distortion:**

Distortion at the edges of Images caused by the use of curved lenses in real cameras. These curves cause a bend in light rays that often bend a little too much or too little at the edges of the lenses. This distortion effect makes lines appear more or less curved than they actually are.

It is the most common type of distortion.

**Tangential Distortion:**

Occurs when a camera’s lens is not aligned perfectly parallel to the imaging plane, where the camera film or sensor is. It makes Images look tilted at some objects appear farther away or closer than they actually are.



**Distortion Coefficients and Correction**

These Distortions can be represented by Distortion coefficients that reflect how distorted our images is.

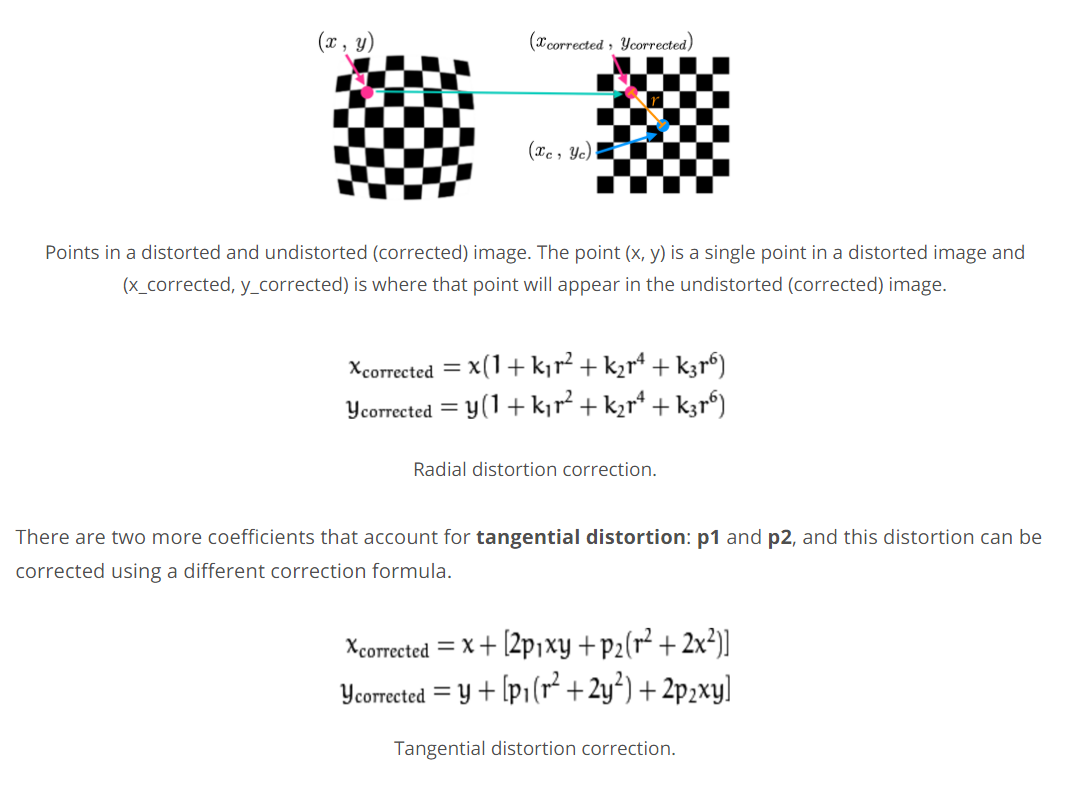
Radial Distortion can be represented by 3 coefficients, k1, k2, and k3.

Note: k3 is required for major radial distortion on things like wide angle lenses, but is generally zero on most regular camera lenses.

Tangential Distortion can be represented by 2 coefficients p1, p2.

Distortion values are typically put in an array (k1 k2 p1 p2 k3)

(x,y) is a point in a distorted image, while r is the known distance between a point in an undistorted image (Xcorrected,Ycorrected) and the center of the image distortion, often the center of the image (Xc,Yc).



To calibrate for distortions, we take an image of a known shape and then calculate the distortion coefficients, which helps us calculate or transform. The known image that is most used is the chess board. Its high contrast pattern helps detect distortions easily.

We can take multiple pictures of a chess board across a flat surface and then we can look at the apparent difference in size and shape of the distorted image to the values they actually are. Then the transform is calculated to undistorted your images.

**Look at the Finding Corners Script for an example.**

Print out the chessboard pattern and stick it to a flat surface. Take 20 to 30 pictures of the chess board from different angles and distances while ensuring they cover the enter field of view.

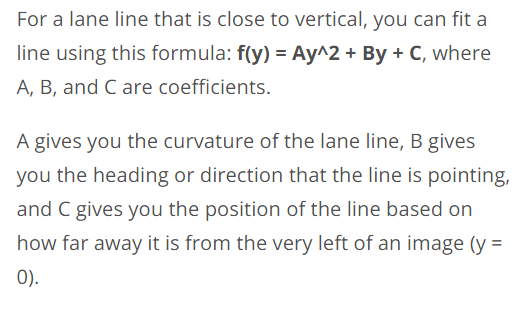
Use these images to calibrate your image

**Calculating Lane Curvature**

Now that our calibration is complete, here are the steps.

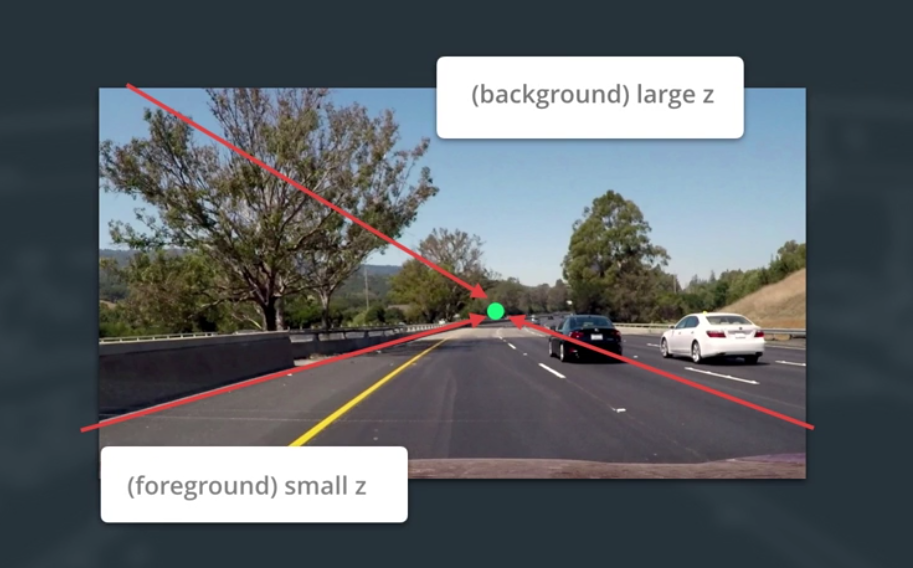
1. Detect lane lines using masking and thresholding
2. Perform a perspective transform to get a Birdseye view of the lane (Lets us fit a polynomial to the lane which is not simple without the perspective transform )
3. We extract the curvature of the lane with the polynomial we find using the points

We fit the 2nd degree polynomial to the lane line in order to gain some information about the curvature.



**Perspective Transform**

Perspective is the phenomina where an object appears smaller the farther it is from a viewpoint and parallel lines seem to converge to a point.



We can characterize perspective by saying in real world co-ordinates (x,y,z) the greater the z value from the camera, the smaller it will appear in the 2D image.

This info is used by the perspective transform. It transforms the apparent z co-ordinate of the object points which changes that objects 2D image. It effectively drags and pushes points from the camera to change the apparent perspective.

It maps the points in a given image to different, desired, image points with a new perspective.

The birds-eye view is the perspective we want to transform to calculate the lane curvature. This is because the curve in the lane lines is much more apparent with a top down view. That view shows that both lanes curve in the same manner.

The top down view also helps us match the road information with a map.

The perspective transform involves a similar process as calibration. Instead of mapping object points to image points, we map the points in a given image to different desired image points with a new perspective.

In openCV we have a getPerspectiveTransform method that gives us the transform with the input of the source and destination points. 4 Points that defined a rectangle on a plane are enough to transform from one perspective to another (Look at Perspective Code)

Now we want to detect the edges of the lane lines in our Image.

In the intro lesson, we used Canny Edge Detection to find our edges. However it gave us a lot of edges that we discarded.

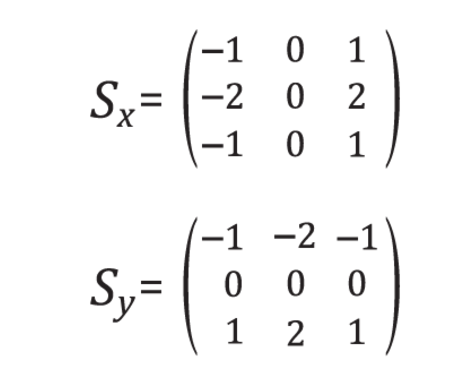
To make a better edge detector for lane lines, we can take advantage of the fact that lane lines are close vertical. We can use the calculations of the gradient of the image that is a part of canny detection in a way to detect steeper edges that are more likely to be a lane line in the first place.

Taking the derivative of an image involves using the Sobel Operator.

**Sobel Operator**

The Sobel Operator takes the derivative in the x-direction, or the y-direction.

Here are the operators for x and y Sobel. These are the 3x3 operator (kernel is the size of 3, the minimum size). The kernel can be any odd number, and the larger the kernel, the gradient is taken over a larger region, meaning a smoother gradient.



If you had an image with color values rising from left to right, and applied x-direction Sobel, you would get a positive output.

When using the Sobel operator on an image of the road, the derivative in the x-direction emphasizes edges closer to vertical, while the derivative in the y-direction emphasizes edges closer to horizontal.

We can use this information to apply a threshold individually on different parts of the derivative. We can threshold the magnitude of the Sobel in the x or the y individually, we can threshold the whole gradient (which is the square root of the add squares of sobelx and sobely), and we can even threshold the direction, which is the arctan of the magnitude of sobely, and sobelx. (Direction of an image is expressed as an angle from horizontal in the range of –pi/2 to pi/2)

**Color Spaces**

So far we have always went from the input of an RGB image and converted it to gray scale in order to apply the derivative function or the canny edge detection algorithm.

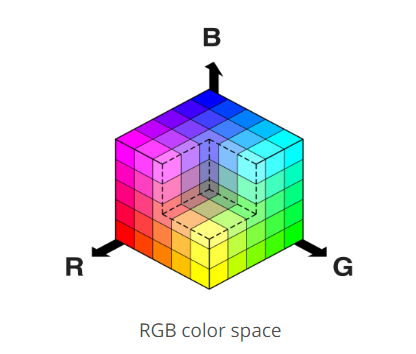
When converting images to gray scale, we lose some valuable information on color variation. When we have a bright patch of concrete and a yellow lane line, if we convert to gray scale, the color information in terms of brightness is lost, and the lane line looks like it disappears.

Different Color spaces gives us more information on an image then grayscale does.

A **Color Space** is a specific organization of colors, it lets us provide a way to categorize colors and represent them in digital images.

**RGB**: Red, Green, Blue color space. It can be thought of as a 3D space, where any color can be represented as a point/coordinate of R, G and B values. Ex. White is (255,255,255)

Due to the ranges of R, G, and B being equivalent, it can be thought of as a point on a 3D Cube that represents a specific Color.



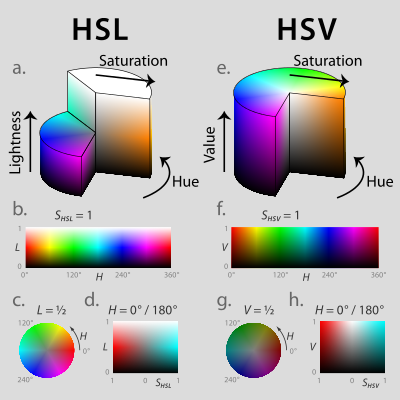
Other color spaces most commonly used in image analysis are **HSV** (Hue, Saturation, Value) and **HLS** (Hue, Lightness, Saturation).

**Hue**: A good way to gain intuition on what Hue is as a value is that it represents color independent of any change in brightness. If you had a basic red paint color, then if you add some white or black to change the brightness, the underlying color is the same and therefore the hue for all those colors is the same.

**Lightness and Value**: These represent different ways to measure the relative lightness or darkness of a color. Ex. A dark red will have a similar hue, but a much lower Value and Lightness than dark red.

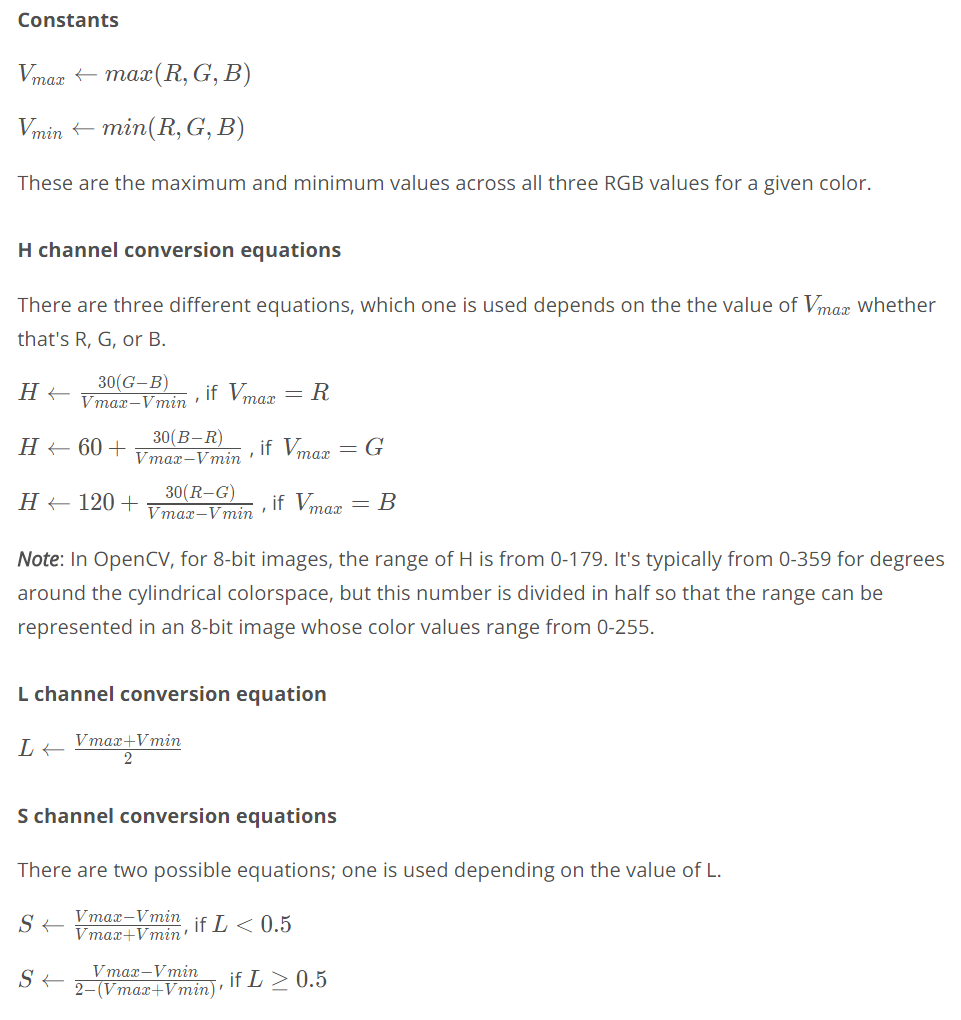
**Saturation**: A measurement of colorfulness. Colors that are lighter and closer to white have lower saturation values, where more intense bright primary colors have high saturation values.

HSL and HSV Color spaces can are represented in Cylindrical Co-ordinates.



Using the HLS space, can help detect lane lines of different colors and under different lighting conditions than grayscale.

Math to Convert Color space from RGB To HLS and HSV values.

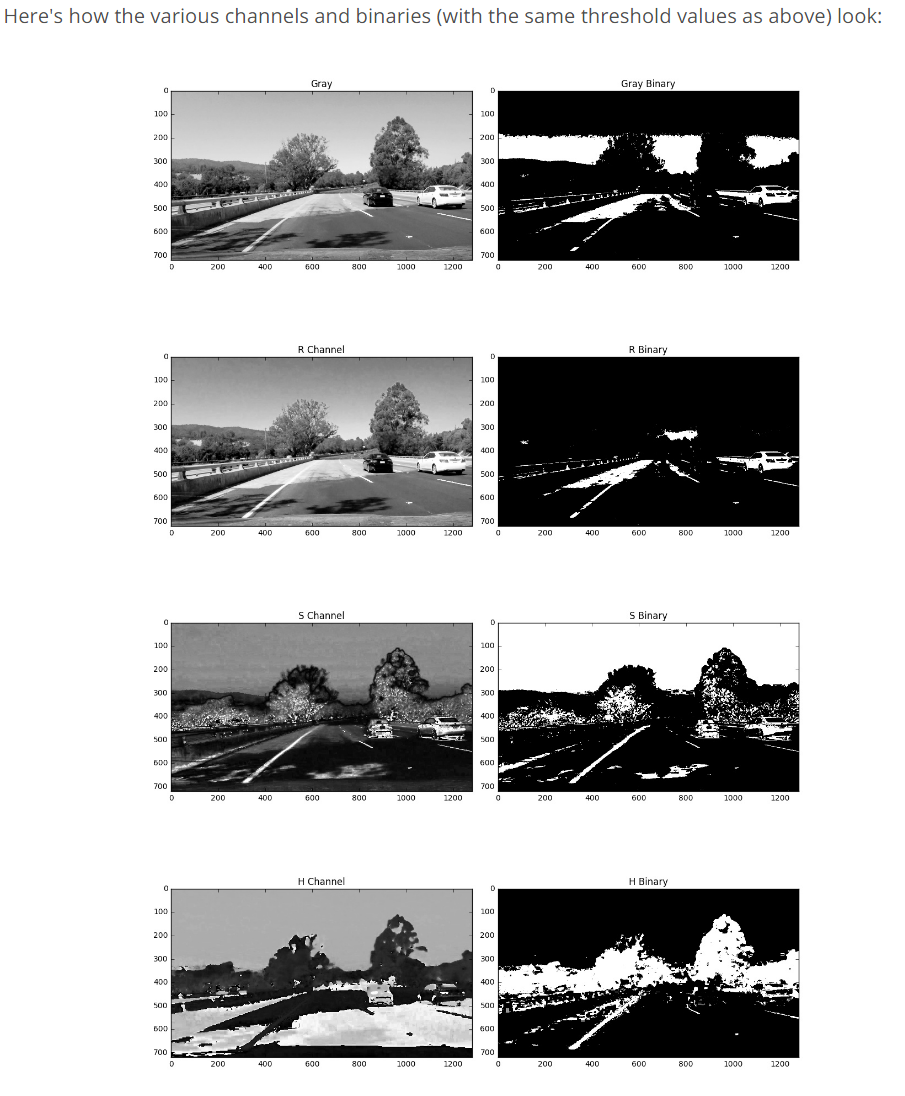


**HLS and Color Thresholds**

So far we approached finding lane lines through a threshold on the derivative of the image.

In the first Lesson, we applied color threshold on the grayscale image.

Now we can see why a color space like HLS can be more robust.



When looking at the same threshold being applied to all these color channels, we can see that the more robust solution to finding lane lines would be to convert the RGB image to a HLS image and then apply a threshold to the S channel.

**Processing Images to detect Lane Lines**

Step 1:

For the project, the first thing you want to do is compute the camera calibration matrix and the distortion coefficients.

Once they are calculated, we can use them to undistort each new frame we want to process.

Step 2:

Apply thresholds on the combinations of color and gradient to generate a binary image where the lane lines are clearly visible.

Step 3:

Perspective Transform the image. To the birds-eye view. We need to find our four source points. For the sake of this project, we can make the assumption that the road is a flat plane, even though that isn’t strictly true. We need to pick 4 points in a trapezoidal shape, similar to what we did with region masking in the first project that would represent the rectangle when looking down on the road from above.

Testing this on an image with straight lane lines is the easiest.

This transform will work on any image (with the assumption that the road is flat and the camera perspective hasn’t changed)

You will know the transform is correct when testing it with new images and seeing that the lane lines should appear parallel with each other in the warped images, whether they are straight or curved.

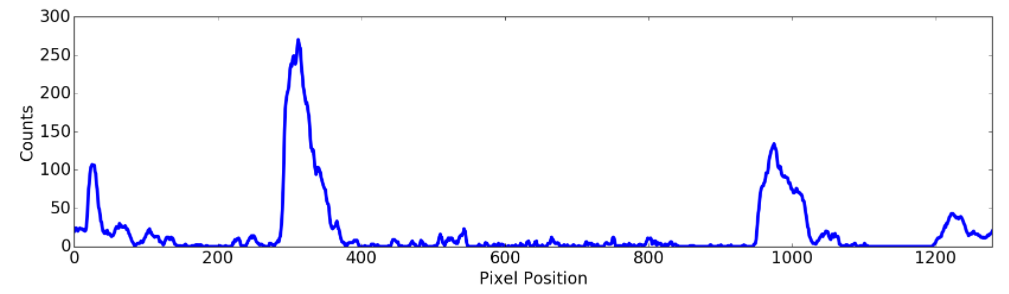
Step 4:

Fit a polynomial to the lane lines.

**Finding the Lane Lines: Peaks in a Histogram**

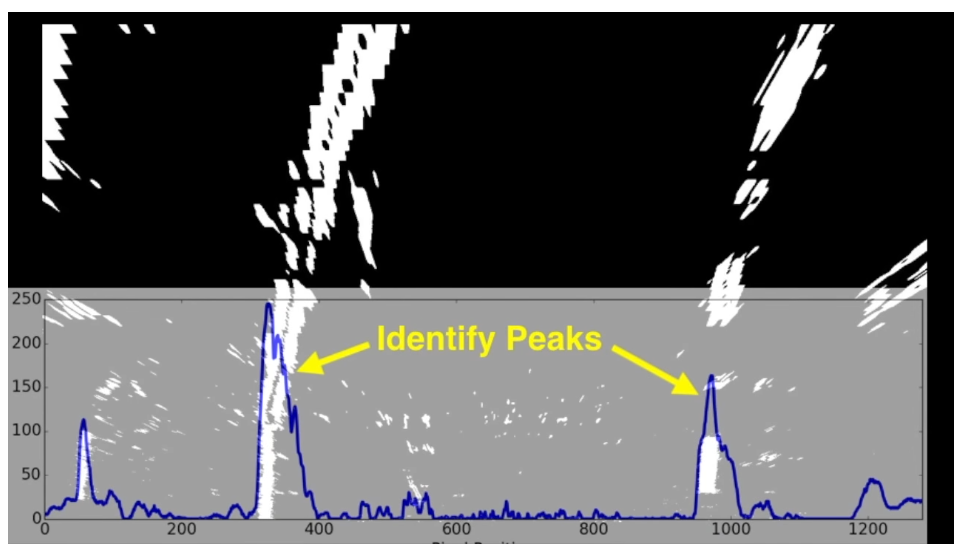
After the process of getting the binary image where the lane lines stand out, we still need to decide explicitly which pixels are part of the lines, and which part belong to the left line and which belong to the right line.

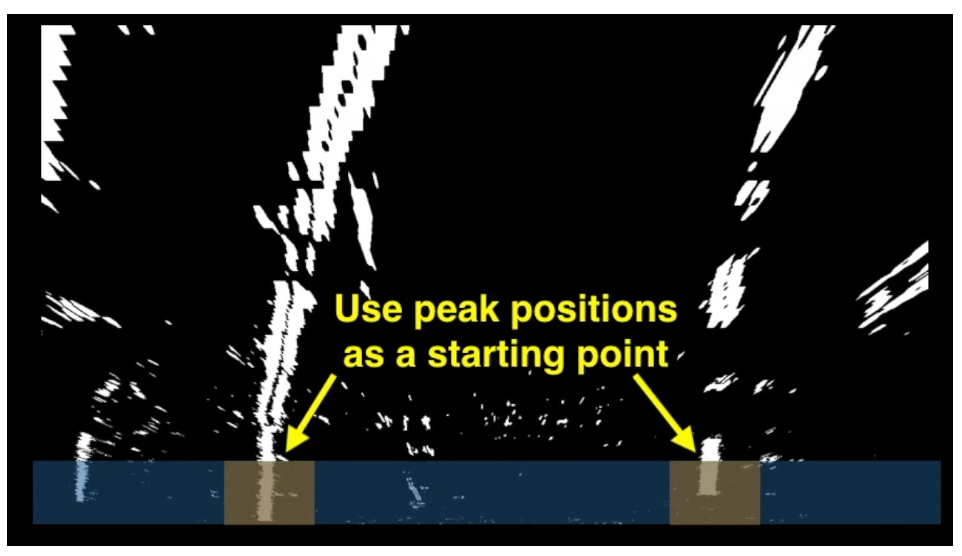
If we look at the peaks in the histogram of the sum of the column values for the bottom half of our image, we can gain some information on the positions of the lane lines.



Because our binary images pixels are either 1 or 0, the histograms most prominent peaks will be good indicators of the x-position of the base of the lane lines.

We can use the x position of these peaks as a starting point of where to search for the lane lines. From that point we can using a sliding window placed around the line centers to find and follow the lines up to the top of the frame.





An example of the code can be seen in the SlidingWindow.py file.

Another approach to do a Sliding window Search is to apply convolution.

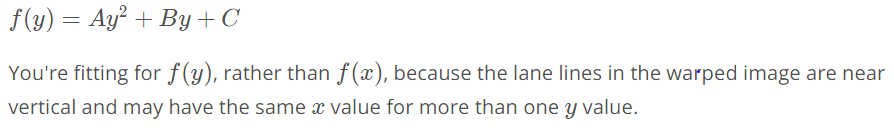
A convolution is a summation of the product of two separate signals, we can use the window template and the vertical slice of the pixel image.

The peak of convolved signal is where there was the highest overlap of pixels, meaning it is the most likely position for the lane marker.

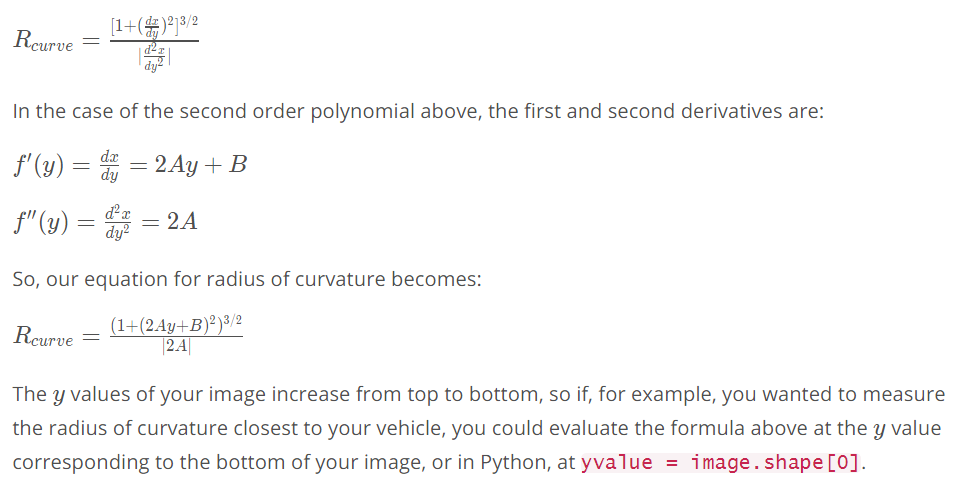
Example is in SlidingWindowConvolution.py

**Measuring Curvature:**

In the previous examples we located the lane line pixels, used there x and y pixel positions to fit a second order polynomial curve.



**The Radius of the Curvature can be found with:**



It is important to understand that values should be translated to real world values rather than pixel values. This involves measuring how long and wide the section of lane is that we’re projecting in our warped image.

Normally, this could be done in detail by measuring the physical lane in the field of view of the camera, but for the project we can make the assumption that the lane is about 30 meters long and 3.7 meters wide.