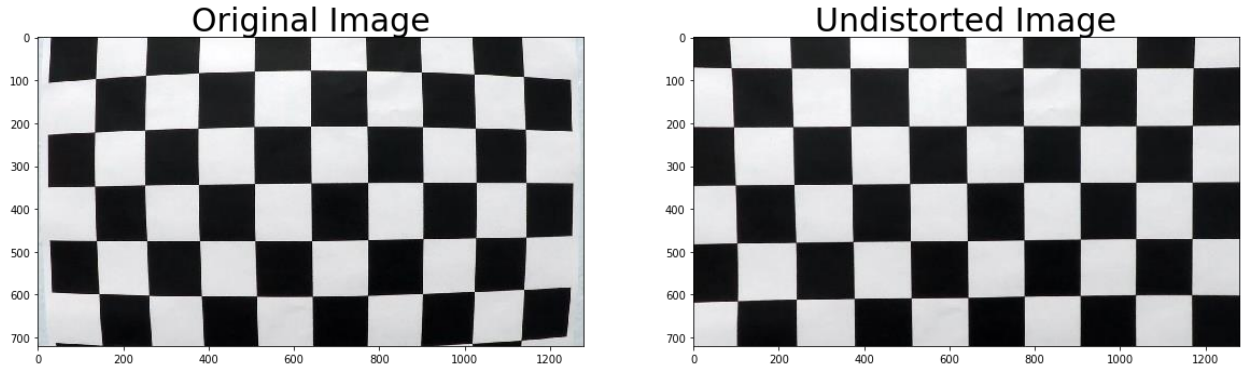
Advanced Lane Detection

This is for project 2, of Udacity's SDC ND. In this lane needs to be marked on road from given video. In the given clip there many variations and hard conditions like Left and right curves on road, road repairs markings with the lane, shadow of tress on lane markings, different type of roads (tar to concrete and back to tar) and tire marks on road. To accomplish this a pipeline with below steps is made.

1. Camera Calibration
2. Image transformation from front-view to top-down view
3. Apply color transformations and filters to above image to detect lanes on image.
4. Check activated pixels for left and right lanes and using sliding window get list of activated pixels.
5. Fit a curve for activated pixels and fill them with color for detected lane.
6. Undo the top-down view to front view to get back lanes on original image.
7. Break view into frames and apply above 1-6 steps in loop to get lane marked.

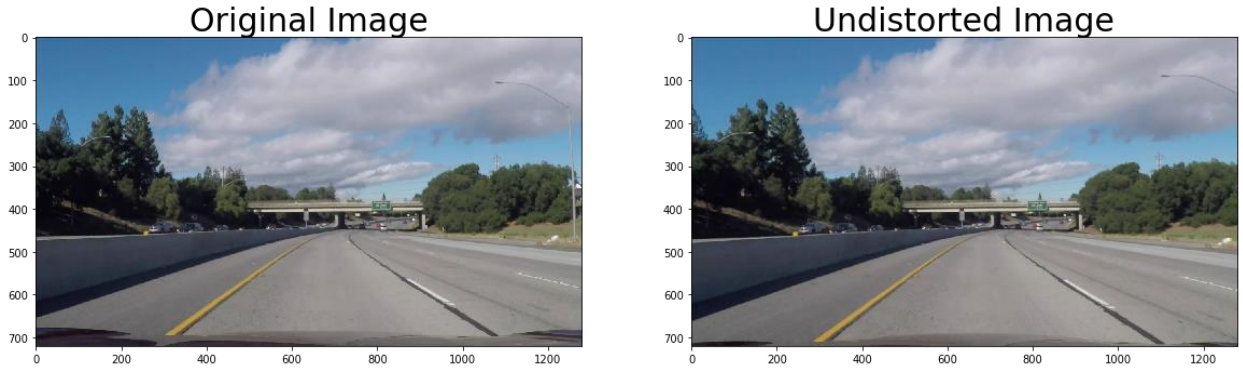
1: Camera Calibration

above in 3-D while a camera captures same in 2-D. This creates discrepancies in actual object in world and how it looks in image. To undo the effects of it, need to find a transformation matrix and distortion matrix. These can be computed if can map a real-world object with its image from camera. For this generally a chess board is used as we know the number of corners and edges in it. For this we call "`cv2.findChessboardCorners`" with parameters as given image and number of corners in it. It returns positions of corners in given image. With this we can call "`cv2.calibrateCamera`" to get required camera matrix and distortion matrix. These matrices is then fed to "`cv2.undistort`" with given image and we get back our original world-image. Using this we get images like below.



From above images we can clearly see how a proper black and white square in chess board were captured by camera (left image) and how using camera calibration we can get back to original image. Generally, for good estimation of original image we should have about 20 images for calibration.

When same is applied on give test image we see how our view changes for side lines and edges. There is some information loss well but that does not affect any area we are interested in. (no lanes are trimmed)



This is done is **calibrate_camera_params** which processes a list of images which have well defined corners. This stores camera params in pickle file which can be loaded when required.

2: Image transformation from front-view to top-down view.

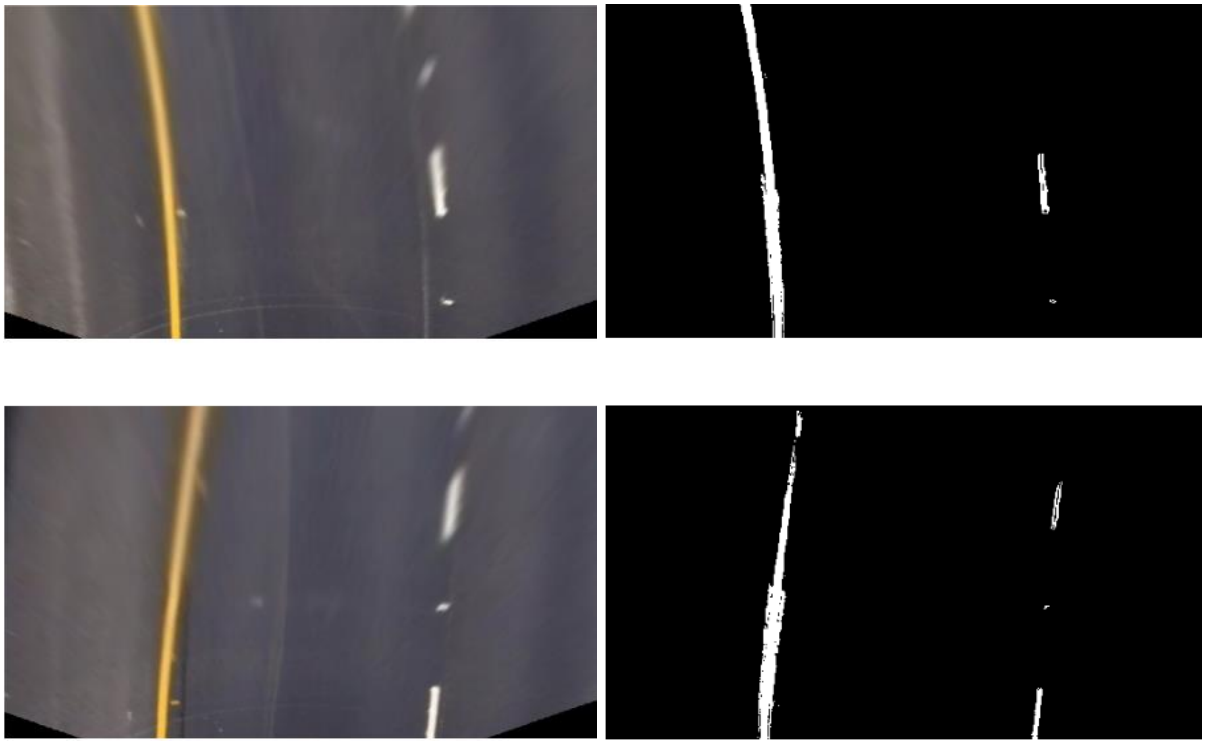
Once we have got back our original world-image, we need to transform this image to get a better look at lanes. In front view (default image capture) we see a lot of data like near by cars, traffic sings and things which are roadside. To get an image which is rich in content which for which we are looking for we need to crop interested area and transform it from front-view to top-down view. This will eliminate objects which are not useful for us in given task. The region of interest in image is part which has lane markings mostly current lanes. The points for that (220,720), (1110, 720), (570, 470), (722, 470) {lower left, lower right, upper left and upper right}. These points need to be transformed to (320,720), (920, 720), (320, 1), (920, 1) {same order as earlier} to get required view. After this transformation image looks like below. On left we have given image and on right it is transformed to top-down view.

This is done in “wrap_top_down_image” transforms points given on image to a formation which is top-down view of required segment. This also saves a required params to pickle file which can be loaded when required.

3: Apply color transformations and filters to detect lanes.

Once we have got an image with just with required lanes, we can use color transform to get a binary image with lanes pixels activated. We perceive world in RGB color format which is good for us but when using images, we have a flexibility to use many other color formats like HLS/HSV/Lab/YUV and many others. The main reason we need to transform from RGB to other format is that show/clouds and intensity of light changes the color in RGB format but in other color formats the original color is not affected by this and we can detected lanes even when a tree casts it shadow on road or there are a lot tyre marking on road. In this we used a hybrid of color schemes to detect lanes. Color images have 3 planes one for each primary color. From RGB format we have kept Red color and selected pixels with values in 200 to 255 range. From HLS (Hue, Lightness, Saturation) color format we have used L and S planes. L plane is used to find edges in given image and S is used to detected color irrespective of light conditions.

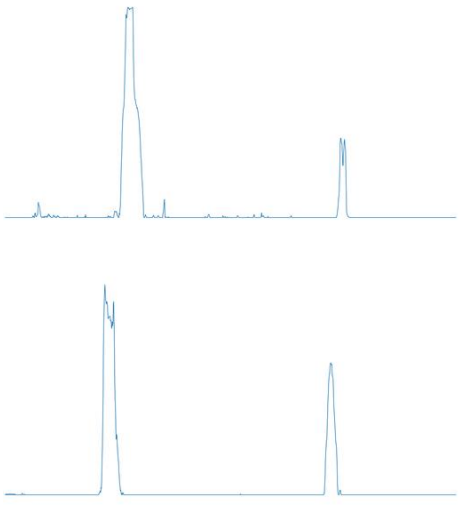
To find edges in L-plane we have used Sobel filter in X direction. This helps us to get gradients in Lightness plane. On S plane we have just filtered all pixels with value above 100. Now we have 3 planes of images and we can combine them to get a final image. To make sure all get equal say in final image we need to normalize Lightness plane images as R and S are already in normalized. We can take a majority vote for activating a pixel in processed image. To convert it to Black and white format we can copy this image to all planes and then multiply it by 255 so that all values are either 0’s and 1’s. After this step we get images like below. In this we can see lanes are marked properly processed image.



This is done in “get_sobel”, calculates Sobel transformation on L channel of HLS in X direction and convert it to binary.

4: Check activated pixels

Once we get a binary image like above, we need to find where are lanes on this. To get lanes we plot a histogram of activated pixels in X-direction. We will get a peak in histograms in X values where we have lanes in image. Histograms looks like below when plotted in X.

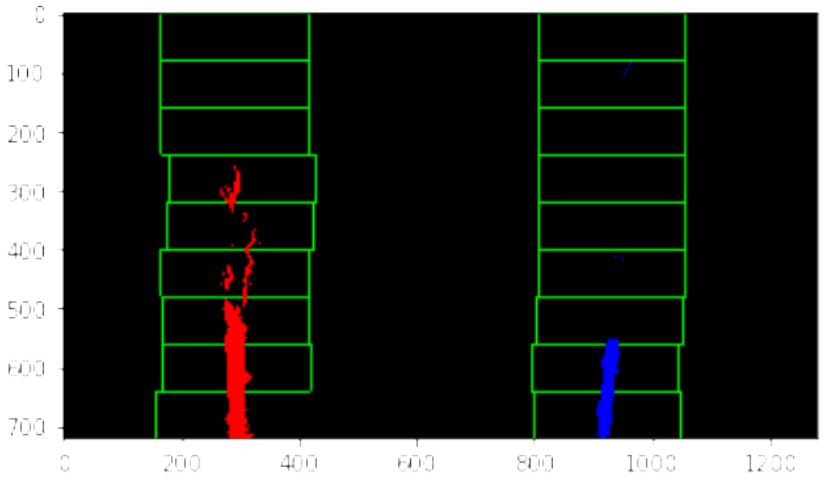


From above plot we see peaks when lane is found in processed image.

This is implement in “image_hist” which takes a binary image and return a histogram for it.

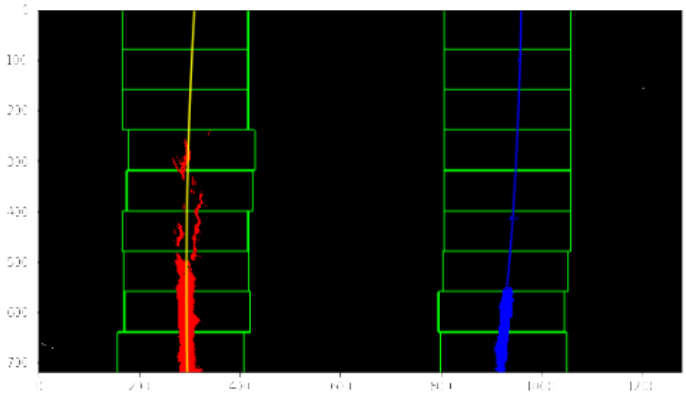
5: Fit a curve for activated pixels

Now comes the most interesting part of project. In processed image we need to search for pixels which are activated and part of lane. We will start from bottom of image and move above creating and trying to connect lane segments. We will divide images in 9 horizontal segments and 2 vertical segments. Vertical ones will be for left and right lane while horizontal will be for getting curve for lanes. To start we will take mid-points in left and right half of image and draw a estimation rectangle on it. We will check for all activated pixels in this rectangle if number of activated pixels is more than threshold (50) then we will update corner for rectangle else we will still be using old one. This helps us to move in direction of lane curvature. After this our processed image looks like below.



For better view we have used red color for left lane, blue color for right lane and green to draw rectangles.

After this we have got list of activated pixels, so now we can fit a smoothing curve to extrapolate these points on a frame to plot lane markings. Once a smoothing cure is plotted processed image looks like below:



From above we can see a smoothing curve is plotted on detected lanes. Left lane is marked with yellow color and right lane is marked with blue color. Using the curve equations, we can find radius of curvature as well.

In this step need to map distance in pixels to real world distance in KM/miles. For that we need multiply x with 3.7/700 (3.7 meters for every 700-pixel, distance between lanes) y with 30/720 (30 meters for every 720-pixel, length of lane markings).

For this “fit_polynomial” is called with given image. It first transforms image to get binary activated pixels and then pass it to “find_lane_pixels”. This first convert binary image to histogram and then apply sliding window to get list of pixels activated for lane. This function returns list of XY coordinates of lanes. Than “fit_polynomial” fits a curve to coordinates list. It tries to do some fail checks if no curve can be fitted to given list of coordinates. It also calculates the curvature of lanes.

6: Undo the top-down view to front view

After lanes are plotted on image, we can fill area between them green color. Once this done, we have detected and marked lanes in images. We need to undo the top-down view back to front-view. For this we can use inverse of earlier transformation. With that we get a image which has lanes marked in region of original image. After this we add initial image (original no lanes marked) and transformed image (with lanes marked on it) to get lanes in original image. We also add current lane curvature and distance from center of lane when adding these images. Distance is calculated with lane curvature equations in meters. We get below image after these steps.

On left is image after inverse transformation and on right is after it is mapped to original image. Vehicle position and curve is also displayed in each frame.

“`get_vehicle_pos`” is called to get position of vehicle from lane center. “`draw_lane`” is called with coordinates of left and right lane to mark it on processed image. It first marks the area between lanes and then undo the top-down view to normal view. It also adds original image and final processed image into one unit.

7: Use pipeline recessively for each frame.

Use the mentioned steps in loop for each frame of video to get lanes.

Apart from these there are various test functions to for each layer check if data from top till that point is processed properly.

display_iamges_glob	This function is used to display images after some type of processing is done.
display_iamges_list	This function is used to display images after some type of processing is done.
display_hist_list	This function is used to display histograms after some type of processing is done.
display_images_list_lane	Displays top-down view of processed image with lanes marked properly on it.
test_wrap_top_down_image	This function tests wrap_top_down_image and get_sobel with given test images
test_image_hist	This function tests image_hist given test images.
test_sliding_window	This function tests fit_polynomial on given test images.
test_draw_img_lane	This function tests draw_lane and get_vehicle_pos on given test images. This also checks end to end lane markings that is from given RGB image to RGB image with lanes marked on it

Discussion:

1: One to the major drawback for this implementation is that, it is tuned for give set of conditions, if any of given conditions changes to large extent this will not work as threshold params might not be valid in those cases.

2: Currently for each frame lanes are searched from scratch, it does not use previous learned/computed lanes and add on new on top of that.

3: Only one lane is marked instead of all lanes in give figure.

4: This might fail if there is a vehicle in same lane with less distance.

5: This is for good sunny day, in foggy/rainy day this will not be able to find lanes.

6: On sharp or zig-zag turns this might perform as expected and will fail.

7: Need to use some deep learning technique (Lanenet) to make is robust to common failure and handle more extreme conditions.

8: Current pipeline does not exploit parallelism of lane lines, it can be helpful to predict the other lane if we not able to find it or fit a curve to it.