Advanced Lane Finding Project

The goals / steps of this project are the following:

* Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.

* Apply a distortion correction to raw images.

* Use color transforms, gradients, etc., to create a thresholded binary image.

* Apply a perspective transform to rectify binary image ("birds-eye view").

* Detect lane pixels and fit to find the lane boundary.

* Determine the curvature of the lane and vehicle position with respect to center.

* Warp the detected lane boundaries back onto the original image.

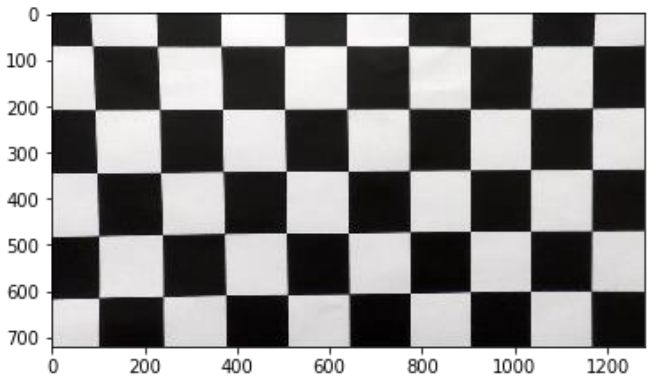
* Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Project References

- ./test_images_output/calibration1.jpg Distortion corrected image
- ./test_images_output/sobel_x.jpg Sobel gradient in x direction
- ./test_images_output/sobel_y.jpg Sobel gradient in y direction
- ./test_images_output/Maginitude_threshold.jpg magnitude threshold
- ./test_images_output/directional_threshold.png directional gradient
- ./test_images_output/s_channel.jpg Thresholded in s channel in an HLS image
- ./test_images_output/combined.jpg Gradients combined
- ./test_images_output/histogram.jpg Histogram for the test1.jpg
- ./test_images_output/Lane_detected.jpg Final image in the project with lane line detected
- project_video_result.mp4 Result video after the lane detection
- harder_challenge_video.mp4 Result video after the lane detection

Project Write up
Rubric Points

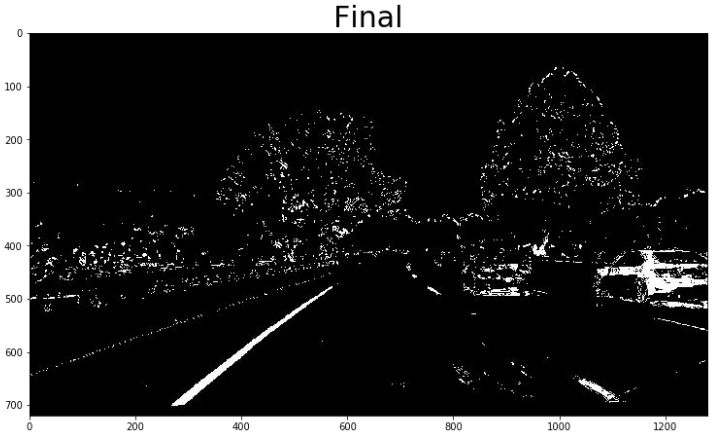
Camera Calibration – created a function which accepts all the chess board images and creates a list of images points. Used the cv2.calibrate function to compute the calibration matrices

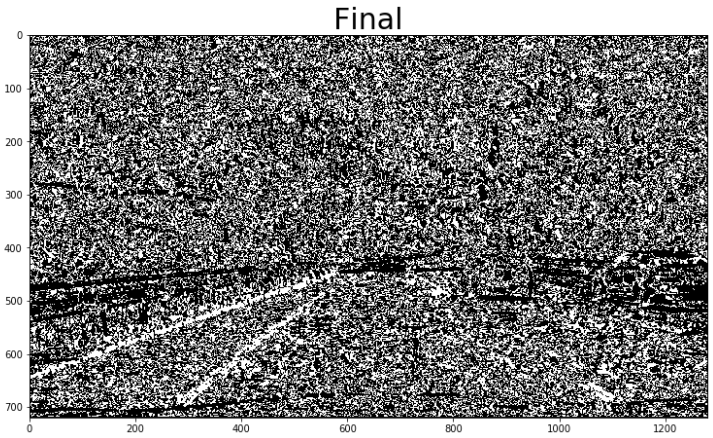


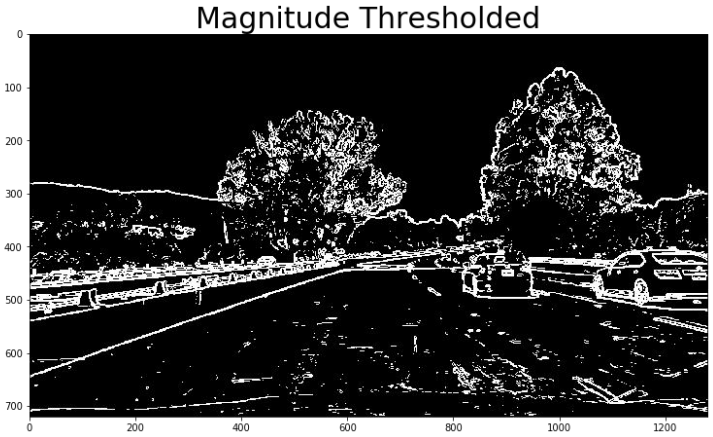
Used the cv2.undistort to un distort the image

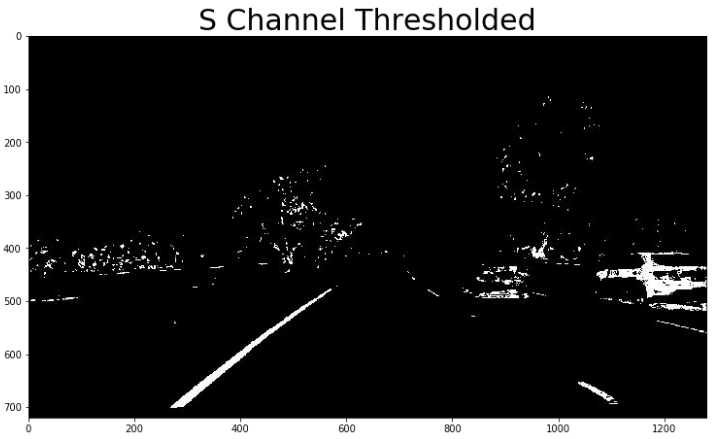
Gradient Calculation:

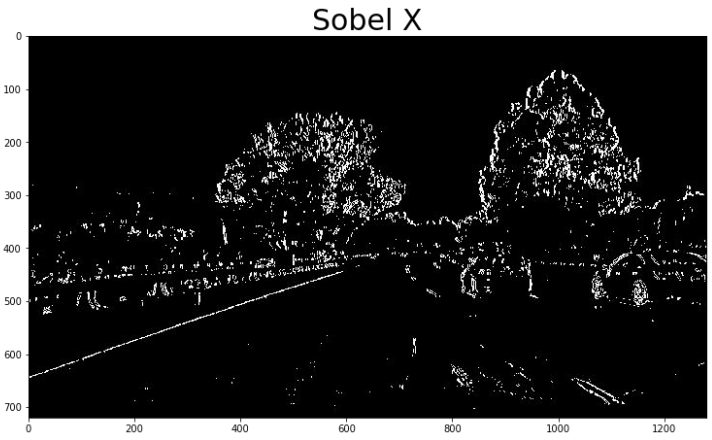
Computed all the different gradients using the sobel and color thresholding and combined all the images using the function Final_Threshold Funtion







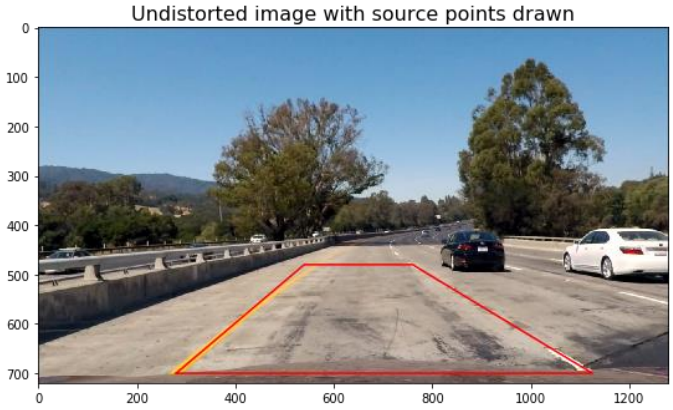


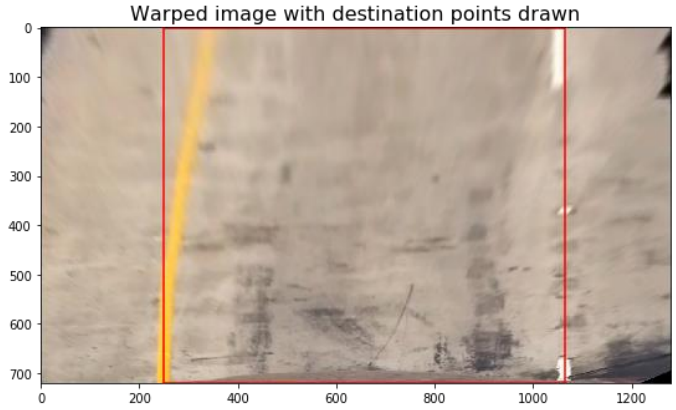


Perspective Transform:

Used the function from the quiz in the lesson to calculate the source and destination points

```
src_coordinates = np.float32(
   [[280,  700],
    [540,  480],
    [760,  480],
    [1125, 700]])
dst_coordinates = np.float32(
   [[250,  720],
    [250,    0],
    [1065,   0],
    [1065, 720]])
```

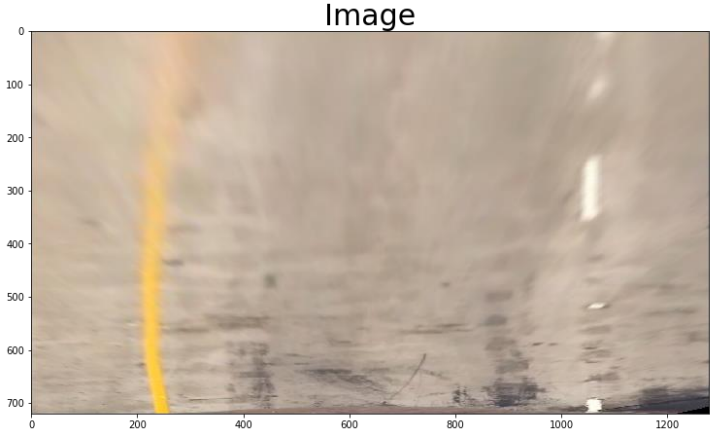




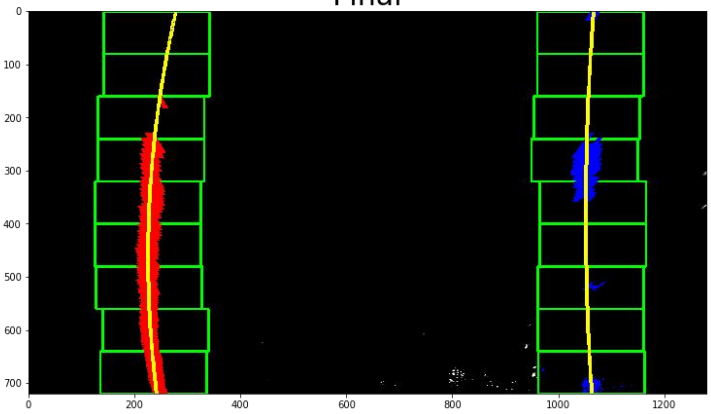
Sliding window algorithm:

Used the sliding window algorithm from the quiz

Image



Final



Final lane detected:

We calculated the lane curvature and points to mark the lanes and plotted the same back to the image

Lane detected with text

Final

Discussion:

The whole algorithm takes a lot of time to calculate over the video the algorithm needs to be improved to run faster and my algorithm has a lot of error at sharp curves and when there is a lot of lighting on the road