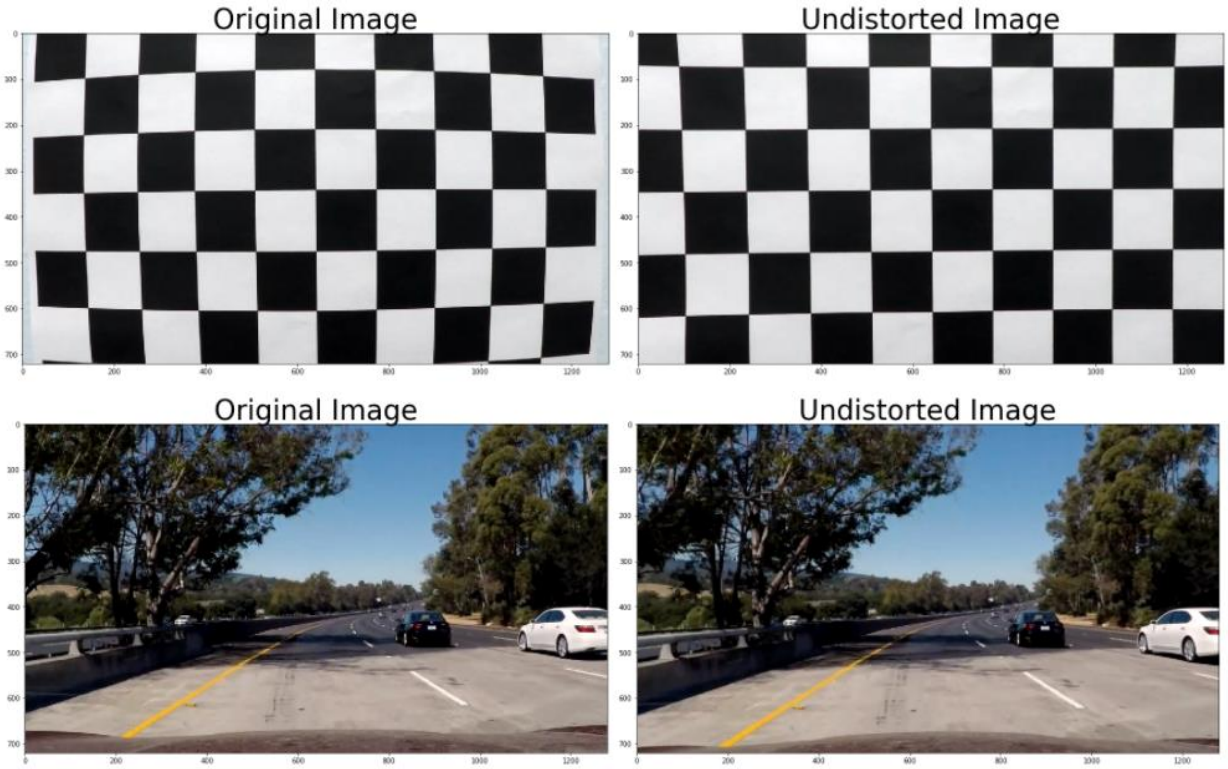
Project 4: Advanced Lane Lines Detection

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.

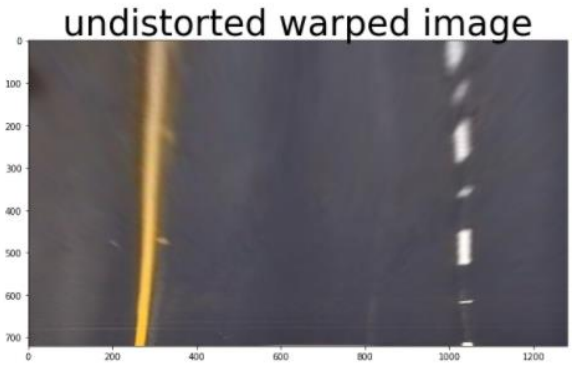
Camera calibration

There are 20 chessboard images used for the camera calibration. Corners of the chessboard are found with the function *cv2.findChessboardCorners().* With the pre-defined object chessboard corners, the camera calibration matrix can be found with the function of *cv2.calibrateCamera().* Then the images can be undistorted.



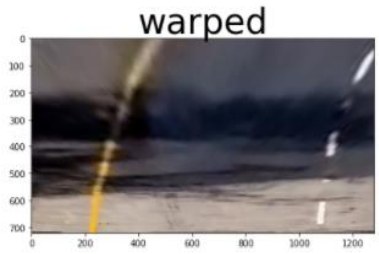
Perspective Transform

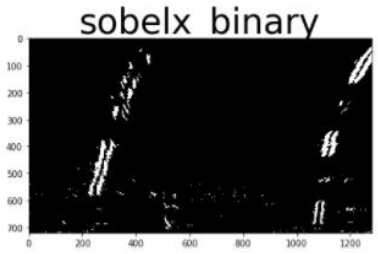
Four points are defined in the undistorted image to enclose the lane lines and those four points are "projected" onto the destination image so that in the destination image, the two lane lines are parallel. With the function of *cv2.getPerspectiveTransform(),* the transform matrix between the destination points and source points can be found. The a *warp()* function is defined to conduct the perspective transform.

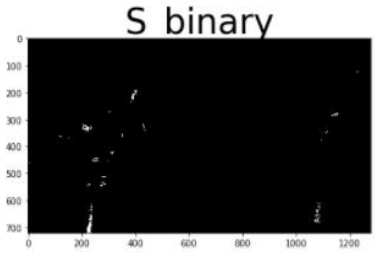


Color thresholding

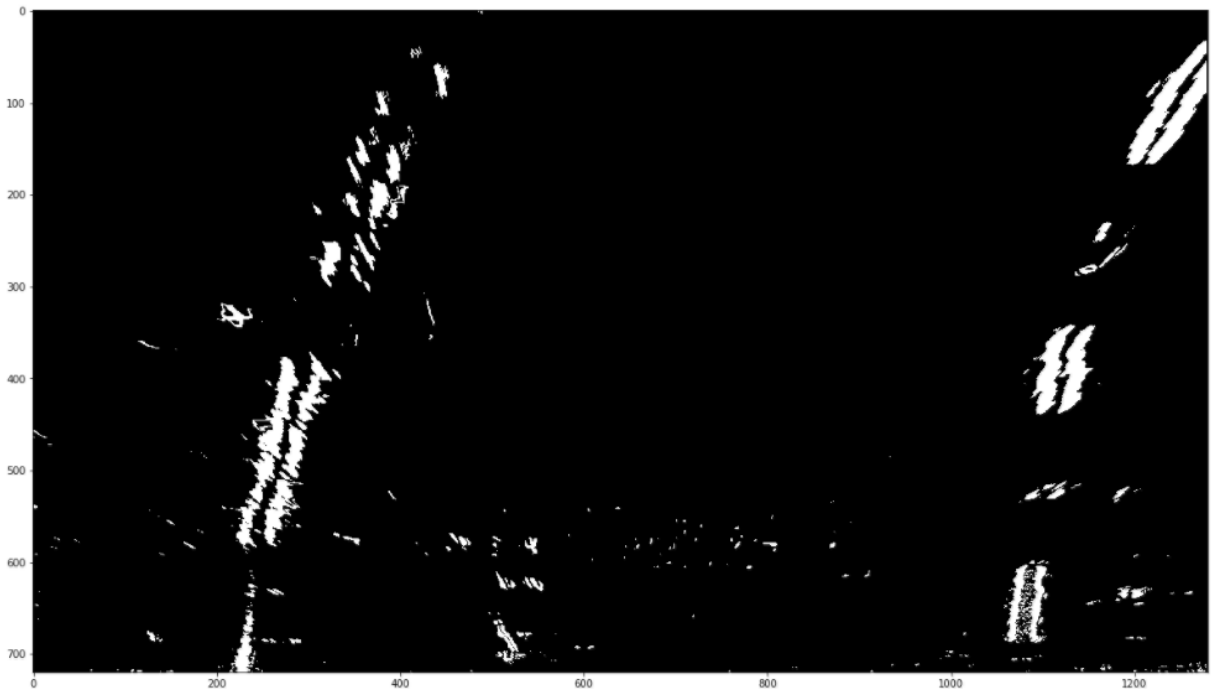
X direction Sobel operation threshold and S color space threshold are applied to the warped images to convert get the lane points. The threshold I use for the Sobelx is (28,255) and (205,255) for S color space.





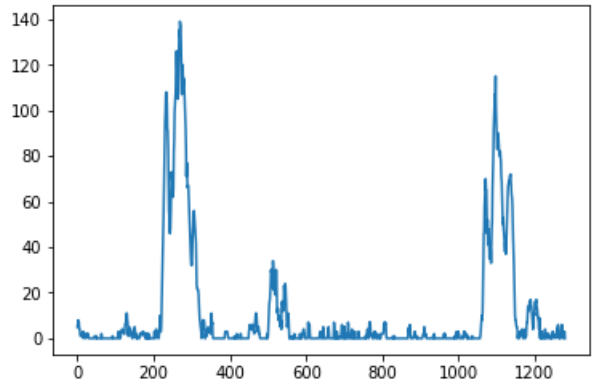


The final binary image of the warped image is:



Detect lane pixels and fit to find the lane boundary

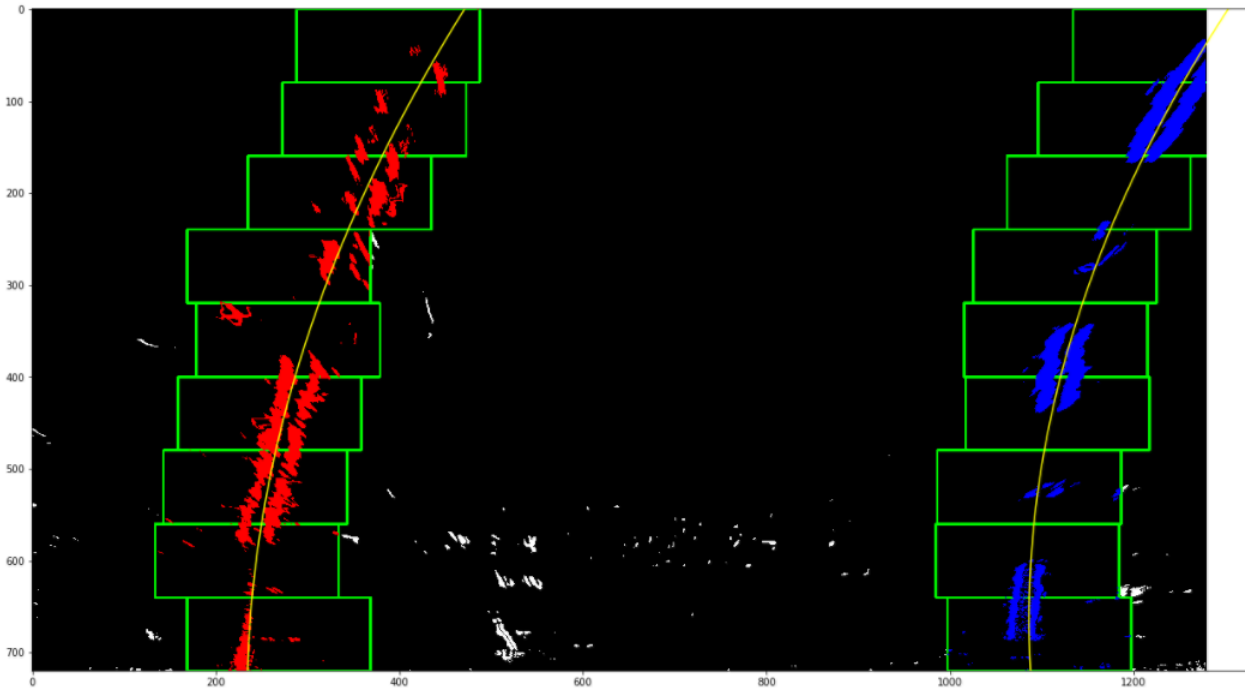
A histogram method is used for the line finding. In the histogram, the white points in the bottom half of the image are counted.



With the histogram, the two peaks in the histogram are the good indicator of the location of the starting points of the two lane lines. Sliding window method is used to find the points of lane lines.

```
# Step through the windows one by one
for window in range(nwindows):
    # Identify window boundaries in x and y (and right and left)
    win_y_low = combined_binary.shape[0] - (window+1)*window_height
    win_y_high = combined_binary.shape[0] - window*window_height
    win_xleft_low = leftx_current - margin
    win_xleft_high = leftx_current + margin
    win_xright_low = rightx_current - margin
    win_xright_high = rightx_current + margin
    # Draw the windows on the visualization image
    cv2.rectangle(out_img,(win_xleft_low,win_y_low),(win_xleft_high,win_y_high),(0,255,0), 2)
    cv2.rectangle(out_img,(win_xright_low,win_y_low),(win_xright_high,win_y_high),(0,255,0), 2)
    # Identify the nonzero pixels in x and y within the window
    good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
    (nonzerox >= win_xleft_low) &  (nonzerox < win_xleft_high)).nonzero()[0]
    good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
    (nonzerox >= win_xright_low) &  (nonzerox < win_xright_high)).nonzero()[0]
    # Append these indices to the lists
    left_lane_inds.append(good_left_inds)
    right_lane_inds.append(good_right_inds)
    # If you found > minpix pixels, recenter next window on their mean position
    if len(good_left_inds) > minpix:
        leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
    if len(good_right_inds) > minpix:
        rightx_current = np.int(np.mean(nonzerox[good_right_inds]))
# Concatenate the arrays of indices
left_lane_inds = np.concatenate(left_lane_inds)
right_lane_inds = np.concatenate(right_lane_inds)
# Extract left and right line pixel positions
leftx = nonzerox[left_lane_inds]
lefty = nonzeroy[left_lane_inds]
rightx = nonzerox[right_lane_inds]
righty = nonzeroy[right_lane_inds]
out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]

# Fit a second order polynomial to each
left_fit = np.polyfit(lefty, leftx, 2)
right_fit = np.polyfit(righty, rightx, 2)
```



With the lane points detected, second order polynomial is used to fit the lane lines.

Determine the curvature and vehicle position respect to the center

With the fitted polynomial, the curvature can be calculated with this function:

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

The curvature radius will then be converted into the world space with the unit of meter by the coefficients in x and y direction, 30/720 meter/pixel and 3.7/850 meter/pixel, respectively. Is assumed the camera is mounted at the center of vehicle, so the distance between the center of the detected lane lines and the center of the image will be regarded as the position of the vehicle after converting to world space.

Warp the detected lane boundaries back onto the original image

The function *cv2.getPerspectiveTransform()* provides the transforming matrix from the warped image to calibrated image.

Summary

The pipeline to process each image is:

- Undistort the image;
- Warp the image into 'bird view'
- Convert the image into binary image with color thresholding and gradient thresholding
- Lane line detection and find the lane lines
- Warp the fitted lane lines back to the original image

A pipeline function is defined to process each image:

```
def process_image(img):
    undistorted=cal_undistort(img, objpoints, imgpoints)
    warped, M, Minv=warp(undistorted)
    combined_binary=bi_image(warped)
    left_fit, right_fit, out_img,left_lane_inds, right_lane_inds=find_lane(combined_binary)
    cur_rad, vehicle_location, dire=get_cur_loc (left_fit, right_fit, combined_binary)
    warp_to_original =warp_back(combined_binary,Minv,w,h, dire, undistorted,left_fit, right_fit)
    add_text(warp_to_original, cur_rad,vehicle_location, dire)
    return warp_to_original
```

Discussion

As is shown in the figure below, the top left corner of the detected lane area goes outbound, that may come from the imperfect of the color threshold and gradient threshold. Given that

condition, the detected lane line points on the top left is not enough, and the top part of the left lane lines is mainly determined by the points on the bottom of the image, thus the polynomial fit is not accurate. To solve this problem, further color threshold and gradient threshold tuning is required.