Finding the Lines

Locate the Lane Lines and Fit a Polynomial

Thresholded and perspective transformed image

You now have a thresholded warped image and you're ready to map out the lane lines! There are many ways you could go about this, but here's one example of how you might do it:

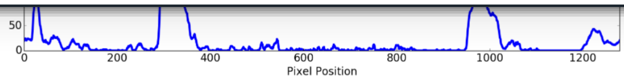
Line Finding Method: Peaks in a Histogram

After applying calibration, thresholding, and a perspective transform to a road image, you should have a binary image where the lane lines stand out clearly. However, you still need to decide explicitly which pixels are part of the lines and which belong to the left line and which belong to the right line.

I first take a **histogram** along all the columns in the *lower half* of the image like this:

```
import numpy as np
histogram = np.sum(img[img.shape[0
plt.plot(histogram)
```

The result looks like this:



Sliding Window

With this histogram I am adding up the pixel values along each column in the image. In my thresholded binary image, pixels are either 0 or 1, so the two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. I can use that as a starting point for where to search for the lines. From that point, I can use a sliding window, placed around the line centers, to find and follow the lines up to the top of the frame.

Here is a short animation showing this method:

Implement Sliding Windows and Fit a Polynomial

Suppose you've got a warped binary image called `binary_warped` and you want to find which "hot" pixels are associated with the lane lines. Here's a basic implementation of the method shown in the animation above. You should think about how you could improve this implementation to make sure you can find the lines as robustly as possible!

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

Mentorship
Get support and stay on track
2

```
histogram = np.sum(binary_warped[b
# Create an output image to draw o
out_img = np.dstack((binary_warped
# Find the peak of the left and ri
# These will be the starting point
midpoint = np.int(histogram.shape[
leftx_base = np.argmax(histogram[:
rightx_base = np.argmax(histogram[

# Choose the number of sliding win
nwindows = 9
# Set height of windows
window_height = np.int(binary_warp
# Identify the x and y positions o
nonzero = binary_warped.nonzero()
nonzeroy = np.array(nonzero[0])
nonzerox = np.array(nonzero[1])
# Current positions to be updated
leftx_current = leftx_base
rightx_current = rightx_base
# Set the width of the windows +/-
margin = 100
# Set minimum number of pixels fou
minpix = 50
# Create empty lists to receive le
left_lane_inds = []
right_lane_inds = []

# Step through the windows one by
for window in range(nwindows):
    # Identify window boundaries i
    win_y_low = binary_warped.shap
    win_y_high = binary_warped.sha
    win_xleft_low = leftx_current
    win_xleft_high = leftx_current
    win_xright_low = rightx_curren
    win_xright_high = rightx_curre
    # Draw the windows on the visu
    cv2.rectangle(out_img,(win_xle
```

```
    (0,255,0), 2)
    # Identify the nonzero pixels
    good_left_inds = ((nonzeroy >=
    (nonzerox >= win_xleft_low) &
    good_right_inds = ((nonzeroy >
    (nonzerox >= win_xright_low) &
    # Append these indices to the
    left_lane_inds.append(good_lef
    right_lane_inds.append(good_ri
    # If you found > minpix pixels
    if len(good_left_inds) > minpi
        leftx_current = np.int(np.
    if len(good_right_inds) > minp
        rightx_current = np.int(np

# Concatenate the arrays of indice
left_lane_inds = np.concatenate(le
right_lane_inds = np.concatenate(r

# Extract left and right line pixe
leftx = nonzerox[left_lane_inds]
lefty = nonzeroy[left_lane_inds]
rightx = nonzerox[right_lane_inds]
righty = nonzeroy[right_lane_inds]

# Fit a second order polynomial to
left_fit = np.polyfit(lefty, leftx
right_fit = np.polyfit(righty, rig
```

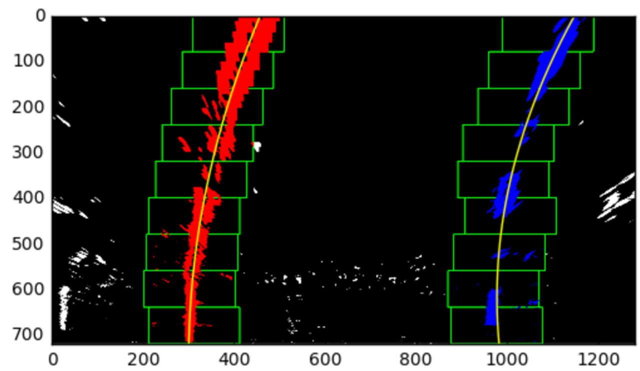
Visualization

At this point, you're done! But here is how you can visualize the result as well:

```
# Generate x and y values for plot
ploty = np.linspace(0, binary_warp
left_fitx = left_fit[0]*ploty**2 +
right_fitx = right_fit[0]*ploty**2
```

```
out_img[nonzeroy[right_lane_inds],
plt.imshow(out_img)
plt.plot(left_fitx, ploty, color='
plt.plot(right_fitx, ploty, color=
plt.xlim(0, 1280)
plt.ylim(720, 0)
```

The output should look something like this:



Skip the sliding windows step once you know where the lines are

Now you know where the lines are you have a fit! In the next frame of video you don't need to do a blind search again, but instead you can just search in a margin around the previous line position like this:

```
# Assume you now have a new warped
# from the next frame of video (al
# It's now much easier to find lin
nonzero = binary_warped.nonzero()
nonzeroy = np.array(nonzero[0])
nonzerox = np.array(nonzero[1])
margin = 100
left_lane_inds = ((nonzerox > (lef
left_fit[2] - margin)) & (nonzerox
left_fit[1]*nonzeroy + left_fit[2]
```

Mentorship
Get support and stay on track
2

Project:
Advanced Lane Finding

Finding the Lines

```
right_fit[2] - margin)) & (nonzero
right_fit[1]*nonzeroy + right_fit[

# Again, extract left and right li
leftx = nonzerox[left_lane_inds]
lefty = nonzeroy[left_lane_inds]
rightx = nonzerox[right_lane_inds]
righty = nonzeroy[right_lane_inds]
# Fit a second order polynomial to
left_fit = np.polyfit(lefty, leftx
right_fit = np.polyfit(righty, rig
# Generate x and y values for plot
ploty = np.linspace(0, binary_warp
left_fitx = left_fit[0]*ploty**2 +
right_fitx = right_fit[0]*ploty**2
```

And you're done! But let's visualize the result here as well

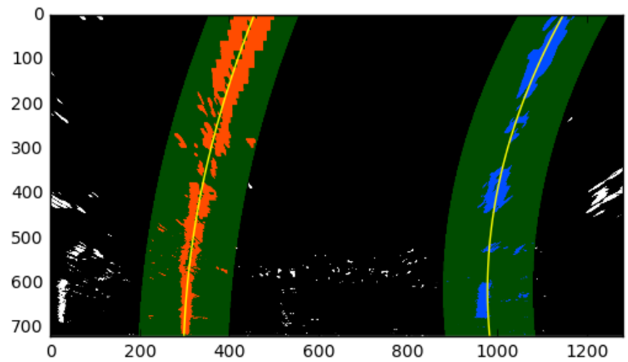
```
# Create an image to draw on and a
out_img = np.dstack((binary_warped
window_img = np.zeros_like(out_img
# Color in left and right line pix
out_img[nonzeroy[left_lane_inds],
out_img[nonzeroy[right_lane_inds],

# Generate a polygon to illustrate
# And recast the x and y points in
left_line_window1 = np.array([np.t
left_line_window2 = np.array([np.f
                              plot
left_line_pts = np.hstack((left_li
right_line_window1 = np.array([np.
right_line_window2 = np.array([np.
                              plot
right_line_pts = np.hstack((right_
```

Mentorship
Get support and stay on track 2

```
cv2.fillPoly(window_img, np.int_([
result = cv2.addWeighted(out_img,
plt.imshow(result)
plt.plot(left_fitx, ploty, color='
plt.plot(right_fitx, ploty, color=
plt.xlim(0, 1280)
plt.ylim(720, 0)
```

And the output should look something like this:



The green shaded area shows where we searched for the lines this time. So, once you know where the lines are in one frame of video, you can do a highly targeted search for them in the next frame. This is equivalent to using a customized region of interest for each frame of video, and should help you track the lanes through sharp curves and tricky conditions. If you lose track of the lines, go back to your sliding windows search or other method to rediscover them.