Lesson 15:
Advanced Techniques for Lane Findi...

Mentorship
Get support and stay on track

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

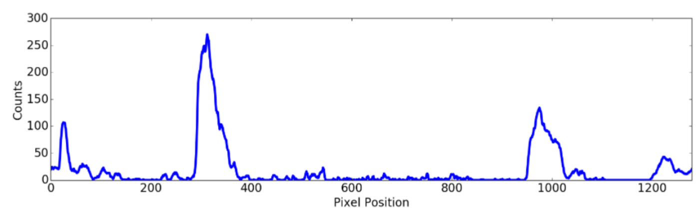
After applying calibration, thresholding, and a perspective transform to a road image, you should have a binary image where the lane lines stand out clearly. However, you still need to decide explicitly which pixels are part of the lines and which belong to the left line and which belong to the right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

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

Finding the Lines

```
:,:], axis=0)
plt.plot(histogram)
```

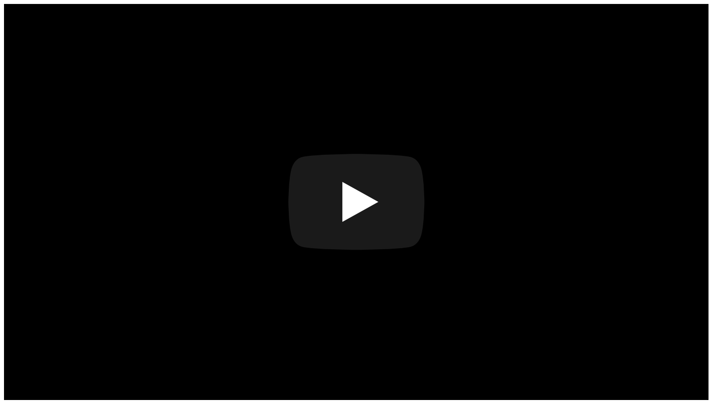
The result looks like this:



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:



Finding the Lines

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!

Lesson 15:
Advanced Techniques for Lane Findi...

Mentorship
Get support and stay on track

Finding the Lines

```
import matplotlib.pyplot as plt

# Assuming you have created a warped bi
nary image called "binary_warped"
# Take a histogram of the bottom half o
f the image
histogram = np.sum(binary_warped[binary
_warped.shape[0]/2:,:], axis=0)
# Create an output image to draw on and
  visualize the result
out_img = np.dstack((binary_warped, bin
ary_warped, binary_warped))*255
# Find the peak of the left and right h
alves of the histogram
# These will be the starting point for
 the left and right lines
midpoint = np.int(histogram.shape[0]/2)
leftx_base = np.argmax(histogram[:midpo
int])
rightx_base = np.argmax(histogram[midpo
int:]) + midpoint

# Choose the number of sliding windows
nwindows = 9
# Set height of windows
window_height = np.int(binary_warped.sh
ape[0]/nwindows)
# Identify the x and y positions of all
 nonzero pixels in the image
nonzero = binary_warped.nonzero()
nonzeroy = np.array(nonzero[0])
nonzerox = np.array(nonzero[1])
# Current positions to be updated for e
ach window
leftx_current = leftx_base
rightx_current = rightx_base
# Set the width of the windows +/- marg
in
margin = 100
# Set minimum number of pixels found to
 recenter window
minpix = 50
# Create empty lists to receive left an
d right lane pixel indices
left_lane_inds = []
right_lane_inds = []

# Step through the windows one by one
for window in range(nwindows):
    # Identify window boundaries in x a
nd y (and right and left)
```

Lesson 15:
Advanced Techniques for Lane Findi...

Mentorship
Get support and stay on track

Finding the Lines

```
 - window*window_height
    win_xleft_low = leftx_current - mar
gin
    win_xleft_high = leftx_current + ma
rgin
    win_xright_low = rightx_current - m
argin
    win_xright_high = rightx_current +
 margin
    # Draw the windows on the visualiza
tion image
    cv2.rectangle(out_img,(win_xleft_lo
w,win_y_low),(win_xleft_high,win_y_hig
h),
    (0,255,0), 2)
    cv2.rectangle(out_img,(win_xright_l
ow,win_y_low),(win_xright_high,win_y_hi
gh),
    (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) &  (non
zerox < win_xleft_high)).nonzero()[0]
    good_right_inds = ((nonzeroy >= win
_y_low) & (nonzeroy < win_y_high) &
    (nonzerox >= win_xright_low) &  (no
nzerox < win_xright_high)).nonzero()[0]
    # Append these indices to the lists
    left_lane_inds.append(good_left_ind
s)
    right_lane_inds.append(good_right_i
nds)
    # If you found > minpix pixels, rec
enter next window on their mean positio
n
    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_la
ne_inds)
right_lane_inds = np.concatenate(right_
lane_inds)
```

Finding the Lines

```
lefty = nonzeroy[left_lane_inds]
rightx = nonzerox[right_lane_inds]
righty = nonzeroy[right_lane_inds]

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

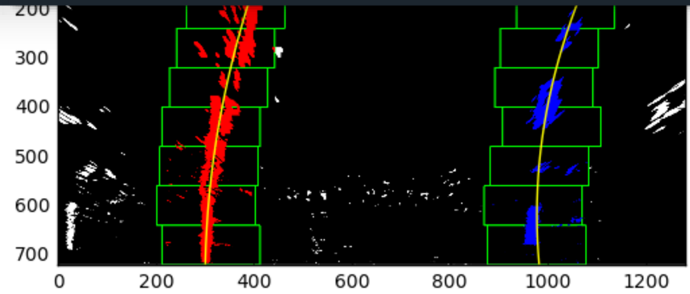
Visualization

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

```
# Generate x and y values for plotting
ploty = np.linspace(0, binary_warped.sh
ape[0]-1, binary_warped.shape[0] )
left_fitx = left_fit[0]*ploty**2 + left
_fit[1]*ploty + left_fit[2]
right_fitx = right_fit[0]*ploty**2 + ri
ght_fit[1]*ploty + right_fit[2]

out_img[nonzeroy[left_lane_inds], nonze
rox[left_lane_inds]] = [255, 0, 0]
out_img[nonzeroy[right_lane_inds], nonz
erox[right_lane_inds]] = [0, 0, 255]
plt.imshow(out_img)
plt.plot(left_fitx, ploty, color='yello
w')
plt.plot(right_fitx, ploty, color='yell
ow')
plt.xlim(0, 1280)
plt.ylim(720, 0)
```

The output should look something like this:

Finding the Lines



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:

Mentorship
Get support and stay on track


```
# from the next frame of video (also ca
lled "binary_warped")
# It's now much easier to find line pix
els!
nonzero = binary_warped.nonzero()
nonzeroy = np.array(nonzero[0])
nonzerox = np.array(nonzero[1])
margin = 100
left_lane_inds = ((nonzerox > (left_fit
[0]*(nonzeroy**2) + left_fit[1]*nonzero
y +
left_fit[2] - margin)) & (nonzerox < (l
eft_fit[0]*(nonzeroy**2) +
left_fit[1]*nonzeroy + left_fit[2] + ma
rgin)))

right_lane_inds = ((nonzerox > (right_f
it[0]*(nonzeroy**2) + right_fit[1]*nonz
eroy +
right_fit[2] - margin)) & (nonzerox <
 (right_fit[0]*(nonzeroy**2) +
right_fit[1]*nonzeroy + right_fit[2] +
 margin)))

# Again, extract left and right line pi
xel positions
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 each
left_fit = np.polyfit(lefty, leftx, 2)
right_fit = np.polyfit(righty, rightx,
2)
# Generate x and y values for plotting
ploty = np.linspace(0, binary_warped.sh
ape[0]-1, binary_warped.shape[0] )
left_fitx = left_fit[0]*ploty**2 + left
_fit[1]*ploty + left_fit[2]
right_fitx = right_fit[0]*ploty**2 + ri
ght_fit[1]*ploty + right_fit[2]
```

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

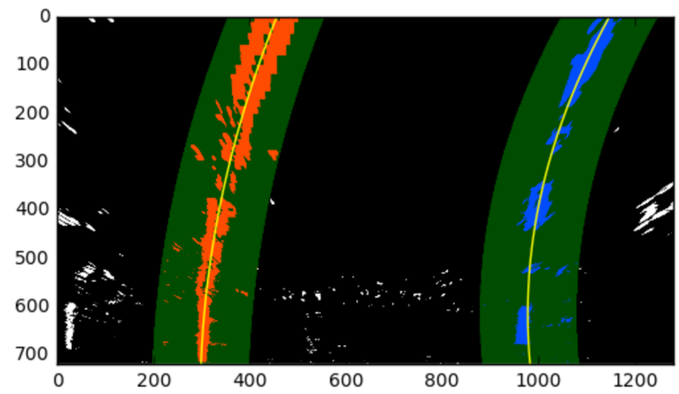
Finding the Lines

```
out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
window_img = np.zeros_like(out_img)
# Color in left and right line pixels
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]

# Generate a polygon to illustrate the search window area
# And recast the x and y points into usable format for cv2.fillPoly()
left_line_window1 = np.array([np.transpose(np.vstack([left_fitx-margin, ploty]))])
left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fitx+margin,
                              ploty])))])
left_line_pts = np.hstack((left_line_window1, left_line_window2))
right_line_window1 = np.array([np.transpose(np.vstack([right_fitx-margin, ploty]))])
right_line_window2 = np.array([np.flipud(np.transpose(np.vstack([right_fitx+margin,
                              ploty])))])
right_line_pts = np.hstack((right_line_window1, right_line_window2))

# Draw the lane onto the warped blank image
cv2.fillPoly(window_img, np.int_([left_line_pts]), (0,255, 0))
cv2.fillPoly(window_img, np.int_([right_line_pts]), (0,255, 0))
result = cv2.addWeighted(out_img, 1, window_img, 0.3, 0)
plt.imshow(result)
plt.plot(left_fitx, ploty, color='yellow')
plt.plot(right_fitx, ploty, color='yellow')
plt.xlim(0, 1280)
plt.ylim(720, 0)
```

Lesson 15:
Advanced Techniques for Lane Findi...

Mentorship
Get support and stay on track

Finding the Lines



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.

NEXT