P4 - Data Analysis

February 28, 2017

1 Advanced Lane Finding Project - Data Analysis & Exploration

The overall goals of the Advanced Lane Line project include:

- 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.

In this data analysis/exploration project, we explore data in each of the steps above and make decision towards our final solution for the project.

1.1 1. Camera Calibration

```
In [1]: import numpy as np
        import cv2
        import glob
        import matplotlib.pyplot as plt
        %matplotlib inline

        def calibrate_camera():

            nx = 9 #enter the number of inside corners in x
            ny = 6 #enter the number of inside corners in y

            objpoints = [] #3D object points
            imgpoints = [] #2D image points

            # prepare object points
```

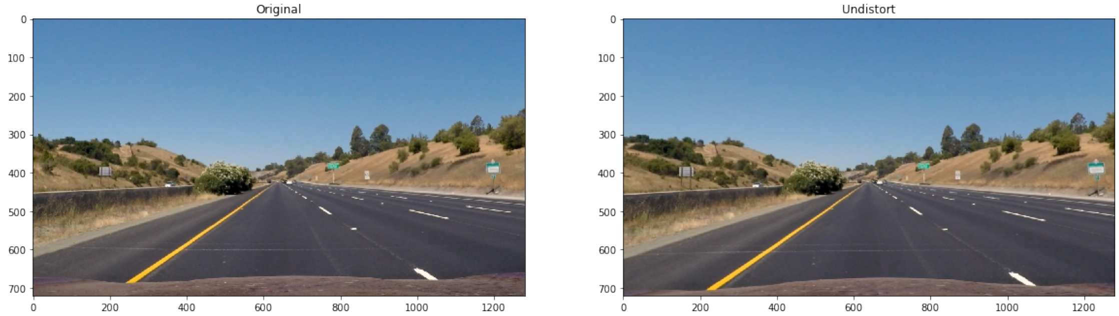
```
    objp = np.zeros((nx * ny, 3), np.float32)
    objp[:,:2] = np.mgrid[0:nx, 0:ny].T.reshape(-1, 2)

    images = glob.glob("./camera_cal/calibration*.jpg")

    for idx, fname in enumerate(images):

        # read image, BGR format in cv2
        img = cv2.imread(fname)

        # Convert to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Find the chessboard corners
        ret, corners = cv2.findChessboardCorners(gray, (nx, ny), None)

        # If found, add to the list of imgpoints and objpoints
        if ret == True:
            objpoints.append(objp)
            imgpoints.append(corners)

    return cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None

# undistort image
def undistort_image(img, mtx, dist):
    return cv2.undistort(img, mtx, dist, None, mtx)

# unwarp image
def unwarp_image(img, src, dst):
    img_height, img_width = img.shape[:2]
    M = cv2.getPerspectiveTransform(src, dst)
    Minv = cv2.getPerspectiveTransform(dst, src)
    warped = cv2.warpPerspective(img, M, (img_width, img_height), flags=cv2
    return warped, M, Minv

print("Calibrating camera ...")
ret, mtx, dist, rvecs, tvecs = calibrate_camera()
print("Done")
```

```
Calibrating camera ...
Done
```

1.2 2. Undistort Image

```
In [2]: %matplotlib inline

        # pick a test image
```

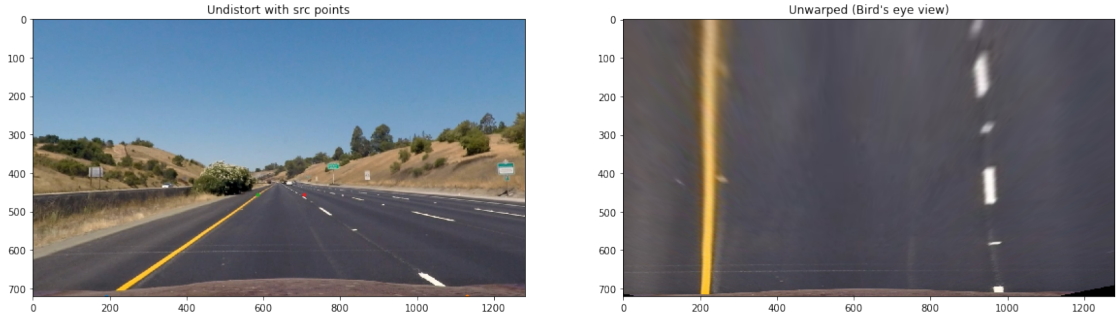
```
img = cv2.imread('./test_images/straight_lines1.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

undist_img = undistort_image(img, mtx, dist)

fig = plt.figure(figsize=(20,10))
rows = 1
cols = 2
plt.subplot(rows, cols, 1)
plt.imshow(img)
plt.title('Original')
plt.subplot(rows, cols, 2)
plt.imshow(undist_img)
plt.title('Undistort')
plt.show()
```


Original

Undistort

1.3 3. Unwarp Image

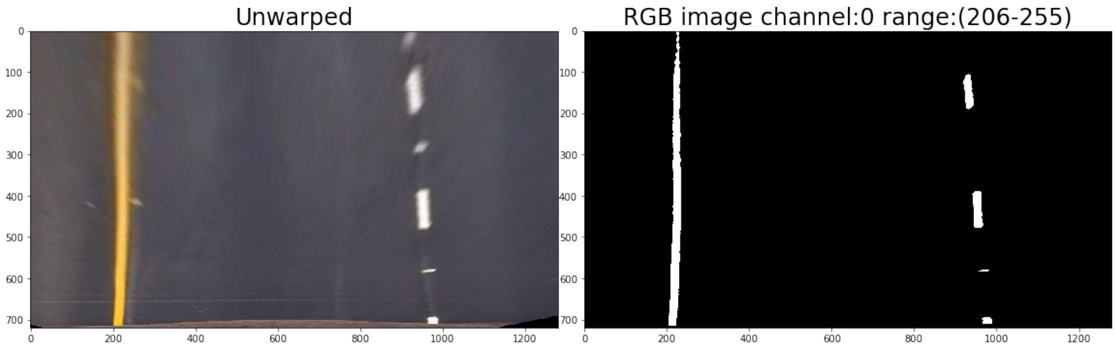
```
In [3]: img_height, img_width = undist_img.shape[:2]
        fig = plt.figure(figsize=(20,10))
        ax = plt.subplot(1,2,1)
        plt.imshow(undist_img)
        plt.title("Undistort with src points")

        #bottom_left = (278, 675)
        #bottom_right = (1044, 675)
        #top_left = (552, 480)
        #top_right = (736, 480)
        bottom_left = (190, 720)
        bottom_right = (1130, 720)
        top_left = (585, 455)
        top_right = (705, 455)

        plt.plot(bottom_left[0], bottom_left[1], '.') # bottom left
        plt.plot(bottom_right[0], bottom_right[1], '.') # bottom right
        plt.plot(top_left[0], top_left[1], '.') # top left
        plt.plot(top_right[0], top_right[1], '.') # top right

        ax.set_ylim([img_height,0])
        ax.set_xlim([0,img_width])

        # source points
        src = np.float32([top_right,
                          bottom_right,
                          bottom_left,
                          top_left])

        # desired destination
        dst_top_left = (200, 0)
        dst_top_right = (1000, 0)
        dst_bottom_right = (1000, 720)
        dst_bottom_left = (200, 720)

        dst = np.float32([dst_top_right,
                          dst_bottom_right,
                          dst_bottom_left,
                          dst_top_left])

        # unwarp image
        unwarp_img, M, Minv = unwarp_image(undist_img, src, dst)

        plt.subplot(1,2,2)
        plt.imshow(unwarp_img)
        plt.title("Unwarped (Bird's eye view)")
        plt.show()
```



1.4 4. Color Channels Thresholding

1.4.1 RGB Colorspace

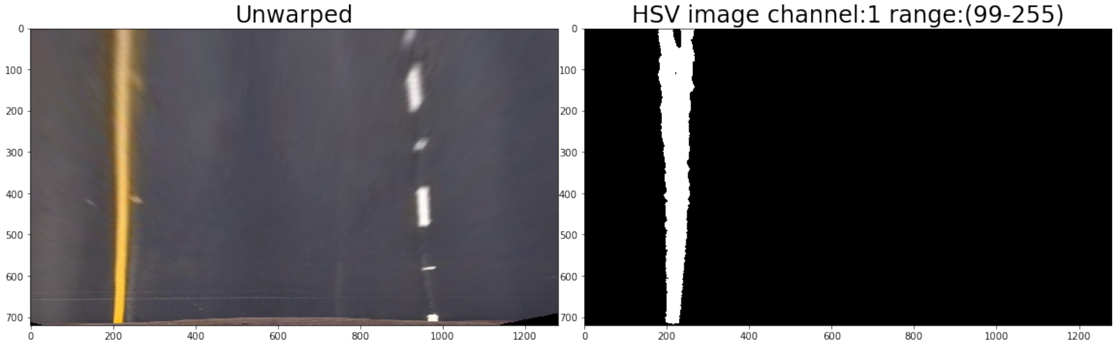
```
In [4]: from ipywidgets import interact, interactive, fixed

        def select(img, ch, thresh):
            channel = img[:,:,ch]
            out = np.zeros_like(channel)
            out[(channel > thresh[0]) & (channel <= thresh[1])] = 1
            return out

        def updateRGB(ch, thresh_min, thresh_max):
            out_img = select(unwarp_img, ch, thresh=(thresh_min, thresh_max))
            f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
            f.subplots_adjust(hspace = .2, wspace=.05)
            ax1.imshow(unwarp_img)
            ax1.set_title('Unwarped', fontsize=24)
            ax2.imshow(out_img, cmap="gray")
            ax2.set_title('RGB image channel:{} range:({}-{})'.format(ch, thresh_mi
            plt.show()

        interact(updateRGB,
                 ch=(0,2),
                 thresh_min=(0,255),
                 thresh_max=(0,255))
```



1.4.2 HSV Colorspace

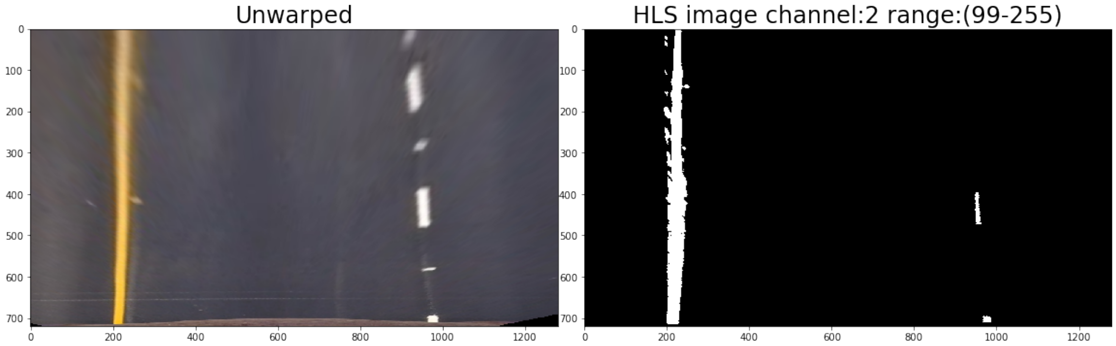
```
In [5]: def updateHSV(ch, thresh_min, thresh_max):
            img = cv2.cvtColor(unwarp_img, cv2.COLOR_RGB2HSV)
            out_img = select(img, ch, thresh=(thresh_min, thresh_max))
            f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
            f.subplots_adjust(hspace = .2, wspace=.05)
            ax1.imshow(unwarp_img)
```

```
    ax1.set_title('Unwarped', fontsize=24)
    ax2.imshow(out_img, cmap="gray")
    ax2.set_title('HSV image channel:{} range:({}-{})'.format(ch, thresh_mi
    plt.show()

interact(updateHSV,
         ch=(0,2),
         thresh_min=(0,255),
         thresh_max=(0,255))
```



1.4.3 HLS Colorspace

```
In [6]: def updateHLS(ch, thresh_min, thresh_max):
            img = cv2.cvtColor(unwarp_img, cv2.COLOR_RGB2HLS)
            out_img = select(img, ch, thresh=(thresh_min, thresh_max))
            f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
            f.subplots_adjust(hspace = .2, wspace=.05)
            ax1.imshow(unwarp_img)
            ax1.set_title('Unwarped', fontsize=24)
            ax2.imshow(out_img, cmap="gray")
            ax2.set_title('HLS image channel:{} range:({}-{})'.format(ch, thresh_mi
            plt.show()

        interact(updateHLS,
                 ch=(0,2),
                 thresh_min=(0,255),
                 thresh_max=(0,255))
```

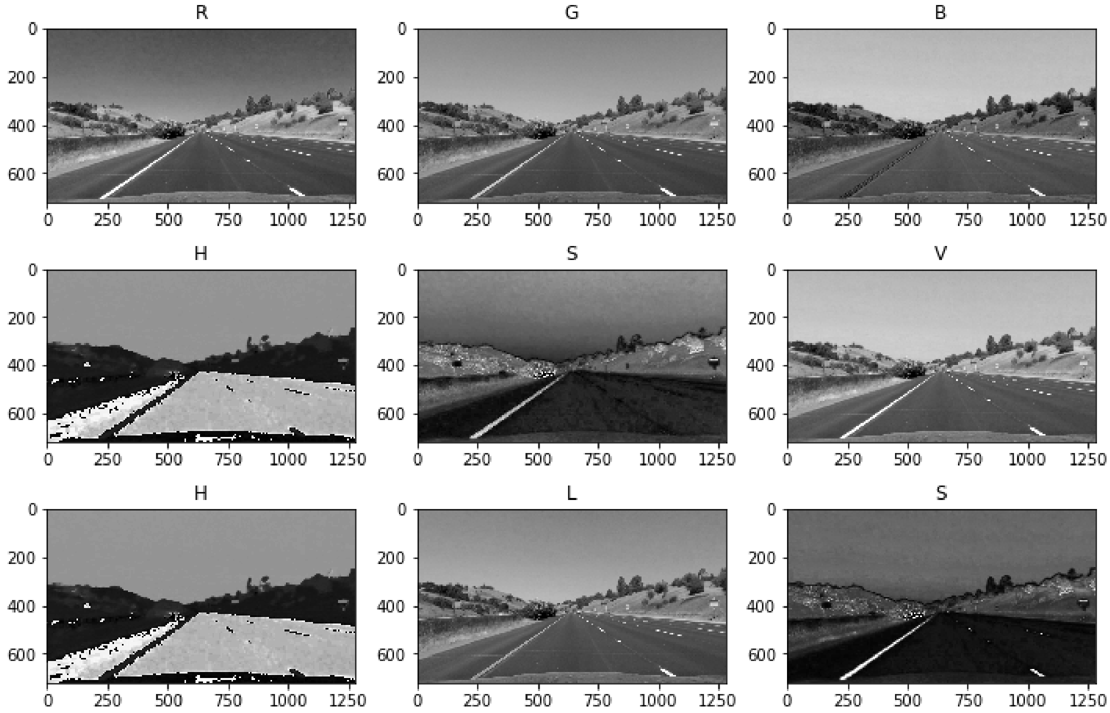

Unwarped

HLS image channel:2 range:(99-255)

1.4.4 Image Comparison (in different color channels)

```
In [7]: def plot_image_color_spaces(img_RGB):
            # setup image channels
            img_R = img_RGB[:,:,0]
            img_G = img_RGB[:,:,1]
            img_B = img_RGB[:,:,2]
            img_HSV = cv2.cvtColor(img_RGB, cv2.COLOR_RGB2HSV)
            img_H = img_HSV[:,:,0]
            img_S = img_HSV[:,:,1]
            img_V = img_HSV[:,:,2]
            img_HLS = cv2.cvtColor(img_RGB, cv2.COLOR_RGB2HLS)
            img_H1 = img_HLS[:,:,0]
            img_L1 = img_HLS[:,:,1]
            img_S1 = img_HLS[:,:,2]
            # plot
            fig = plt.figure(figsize=(12,8))
            ax = plt.subplot(3,3,1)
            plt.imshow(img_R, cmap='gray')
            plt.title('R')
            ax = plt.subplot(3,3,2)
            plt.imshow(img_G, cmap='gray')
            plt.title('G')
            ax = plt.subplot(3,3,3)
            plt.imshow(img_B, cmap='gray')
            plt.title('B')
            ax = plt.subplot(3,3,4)
            plt.imshow(img_H, cmap='gray')
            plt.title('H')
            ax = plt.subplot(3,3,5)
            plt.imshow(img_S, cmap='gray')
            plt.title('S')
            ax = plt.subplot(3,3,6)
            plt.imshow(img_V, cmap='gray')
```

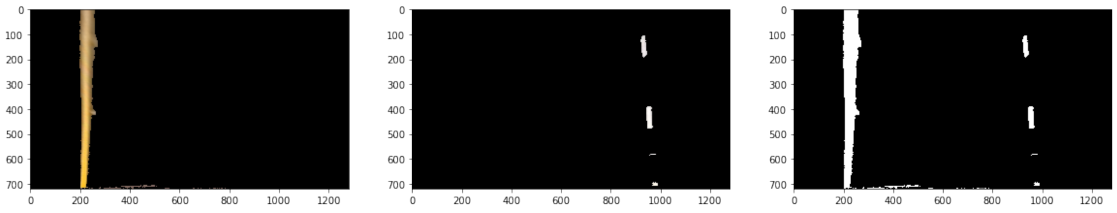
```
    plt.title('V')
    ax = plt.subplot(3,3,7)
    plt.imshow(img_H1, cmap='gray')
    plt.title('H')
    ax = plt.subplot(3,3,8)
    plt.imshow(img_L1, cmap='gray')
    plt.title('L')
    ax = plt.subplot(3,3,9)
    plt.imshow(img_S1, cmap='gray')
    plt.title('S')
    plt.show()

plot_image_color_spaces(undist_img)
```



1.4.5 Yellow and White color lane lines

```
In [8]: def color_mask(img, low, high):
            return cv2.inRange(img, low, high)

        def yellow_lines(img):
            image = np.copy(img)
            hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            yellow_hsv_low  = np.array([ 0,   60,   120])
            yellow_hsv_high = np.array([ 179, 255, 255])
```

```
    mask = color_mask(hsv, yellow_hsv_low, yellow_hsv_high)
    result = cv2.bitwise_and(image, image, mask= mask)
    return result, mask

def white_lines(img):
    image = np.copy(img)
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    white_hsv_low  = np.array([ 0,   0,   205])
    white_hsv_high = np.array([ 255,  255, 255])
    mask = color_mask(hsv, white_hsv_low, white_hsv_high)
    result = cv2.bitwise_and(image, image, mask= mask)
    return result, mask

def white_lines_from_RGB(img):
    image = np.copy(img)
    white_rgb_low  = np.array([ 200,   200,   200])
    white_rgb_high = np.array([ 255,  255, 255])
    mask = color_mask(image, white_rgb_low, white_rgb_high)
    result = cv2.bitwise_and(image, image, mask= mask)
    return result, mask

yellow_lines_binary, yellow_mask = yellow_lines(unwarp_img)
#white_lines_binary, white_mask = white_lines(unwarp_img)
white_lines_binary, white_mask = white_lines_from_RGB(unwarp_img)
lane_mask = cv2.bitwise_or(yellow_mask, white_mask)
fig = plt.figure(figsize=(20,10))
plt.subplot(131)
plt.imshow(yellow_lines_binary, cmap='gray')
plt.subplot(132)
plt.imshow(white_lines_binary, cmap='gray')
plt.subplot(133)
plt.imshow(lane_mask, cmap='gray')
plt.show()
```



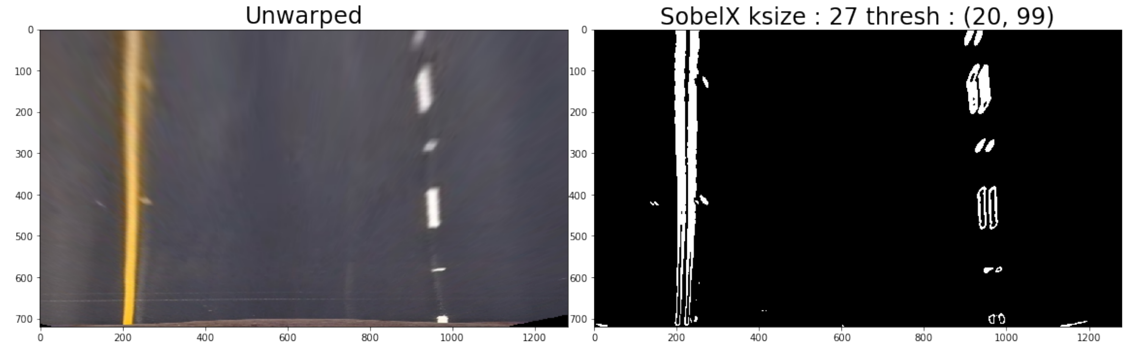
1.5 5. Gradient Thresholding

```
In [9]: def mag_thresh(img, sobel_kernel=3, mag_thresh=(0, 255)):
            gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
```

```
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    gradmag = np.sqrt(sobelx**2 + sobely**2)
    scale_factor = np.max(gradmag)/255
    gradmag = (gradmag/scale_factor).astype(np.uint8)
    binary_output = np.zeros_like(gradmag)
    binary_output[(gradmag >= mag_thresh[0]) & (gradmag <= mag_thresh[1])]
    return binary_output

def abs_sobel_thresh(img, orient='x', sobel_kernel=3, thresh=(0, 255)):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    if orient == 'x':
        abs_sobel = np.absolute(cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sob

    if orient == 'y':
        abs_sobel = np.absolute(cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sob

    scaled_sobel = np.uint8(255*abs_sobel/np.max(abs_sobel))
    binary_output = np.zeros_like(scaled_sobel)
    binary_output[(scaled_sobel >= thresh[0]) & (scaled_sobel <= thresh[1])
    return binary_output

def dir_threshold(img, sobel_kernel=3, thresh=(0, np.pi/2)):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    abs_sobelx = np.absolute(sobelx)
    abs_sobely = np.absolute(sobely)
    absgraddir = np.arctan2(abs_sobely, abs_sobelx)
    binary_output = np.zeros_like(absgraddir)
    binary_output[(absgraddir >= thresh[0]) & (absgraddir <= thresh[1])] =
    return binary_output
```

1.5.1 SobelX Thresholding

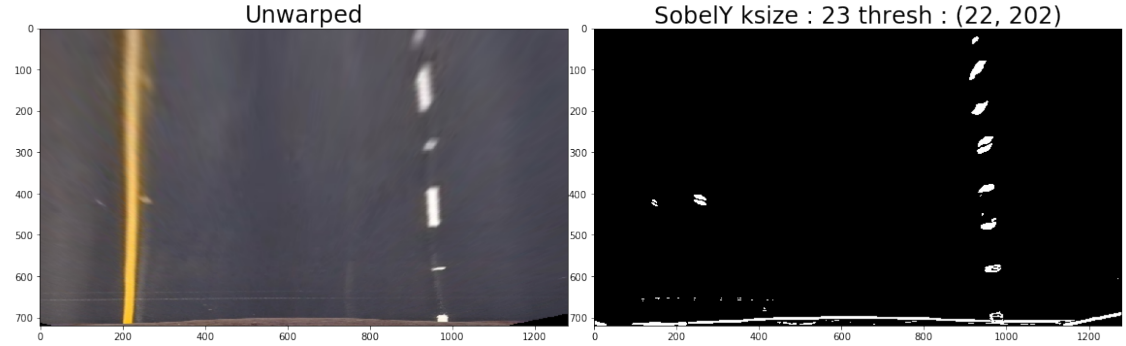
```
In [10]: def updateSobelX(ksize, thresh_min, thresh_max):
             out = abs_sobel_thresh(unwarp_img, orient='x', sobel_kernel=ksize, thr
             f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
             f.subplots_adjust(hspace = .2, wspace=.05)
             ax1.imshow(unwarp_img)
             ax1.set_title('Unwarped', fontsize=24)
             ax2.imshow(out, cmap="gray")
             ax2.set_title('SobelX ksize : {} thresh : ({}, {})'.format(ksize, thre
             plt.show()

         interact(updateSobelX,
                  ksize=(3,31,2),
                  thresh_min=(0,255),
                 thresh_max=(0,255))
```


Unwarped

SobelX ksize : 27 thresh : (20, 99)

1.5.2 SobelY Thresholding

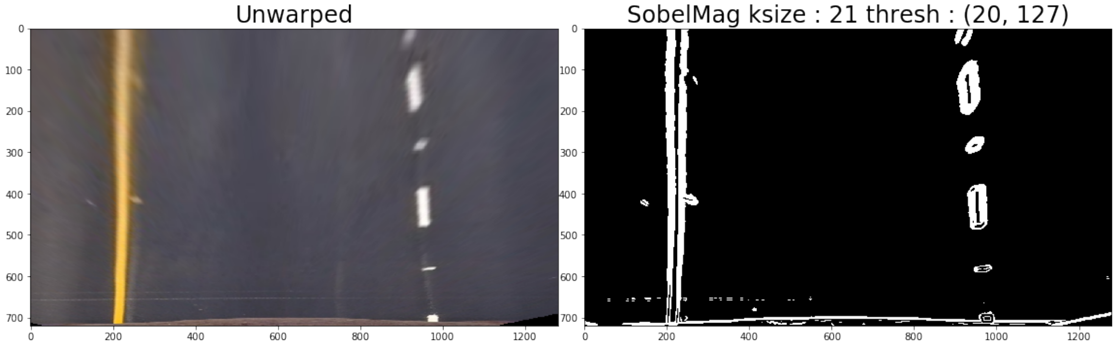
```
In [11]: def updateSobelY(ksize, thresh_min, thresh_max):
             out = abs_sobel_thresh(unwarp_img, orient='y', sobel_kernel=ksize, thr
             f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
             f.subplots_adjust(hspace = .2, wspace=.05)
             ax1.imshow(unwarp_img)
             ax1.set_title('Unwarped', fontsize=24)
             ax2.imshow(out, cmap="gray")
             ax2.set_title('SobelY ksize : {} thresh : ({}, {})'.format(ksize, thre
             plt.show()

         interact(updateSobelY,
                  ksize=(3,31,2),
                  thresh_min=(0,255),
                  thresh_max=(0,255))
```


Unwarped

SobelY ksize : 23 thresh : (22, 202)

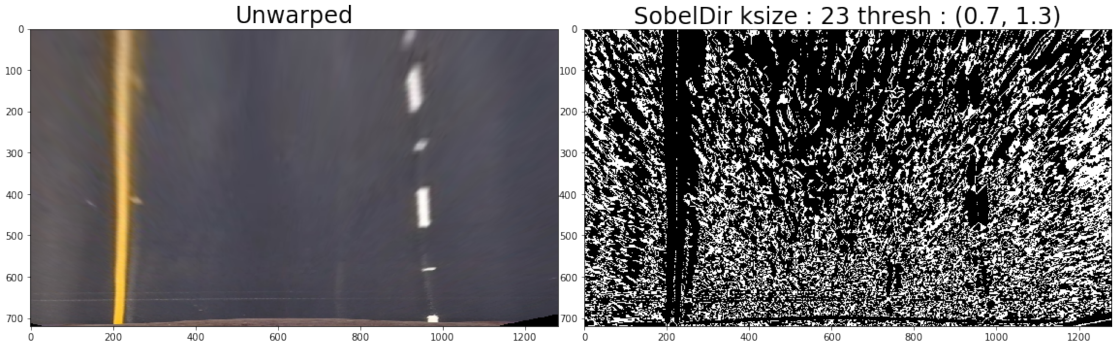
1.5.3 SobelMag Thresholding

```
In [12]: def updateSobelMag(ksize, thresh_min, thresh_max):
             out = mag_thresh(unwarp_img, sobel_kernel=ksize, mag_thresh=(thresh_mi
             f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
             f.subplots_adjust(hspace = .2, wspace=.05)
             ax1.imshow(unwarp_img)
             ax1.set_title('Unwarped', fontsize=24)
             ax2.imshow(out, cmap="gray")
             ax2.set_title('SobelMag ksize : {} thresh : ({}, {})'.format(ksize, th
             plt.show()

         interact(updateSobelMag,
                  ksize=(3,31,2),
                  thresh_min=(0,255),
                  thresh_max=(0,255))
```



1.5.4 Sobel Direction Thresholding

```
In [13]: def updateSobelDir(ksize, thresh_min, thresh_max):
             out = dir_threshold(unwarp_img, sobel_kernel=ksize, thresh=(thresh_mir
             f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
             f.subplots_adjust(hspace = .2, wspace=.05)
             ax1.imshow(unwarp_img)
             ax1.set_title('Unwarped', fontsize=24)
             ax2.imshow(out, cmap="gray")
             ax2.set_title('SobelDir ksize : {} thresh : ({}, {})'.format(ksize, th
             plt.show()

         interact(updateSobelDir,
                  ksize=(3,31,2),
                  thresh_min=(0.0,3.0),
                  thresh_max=(0.0,3.0))
```

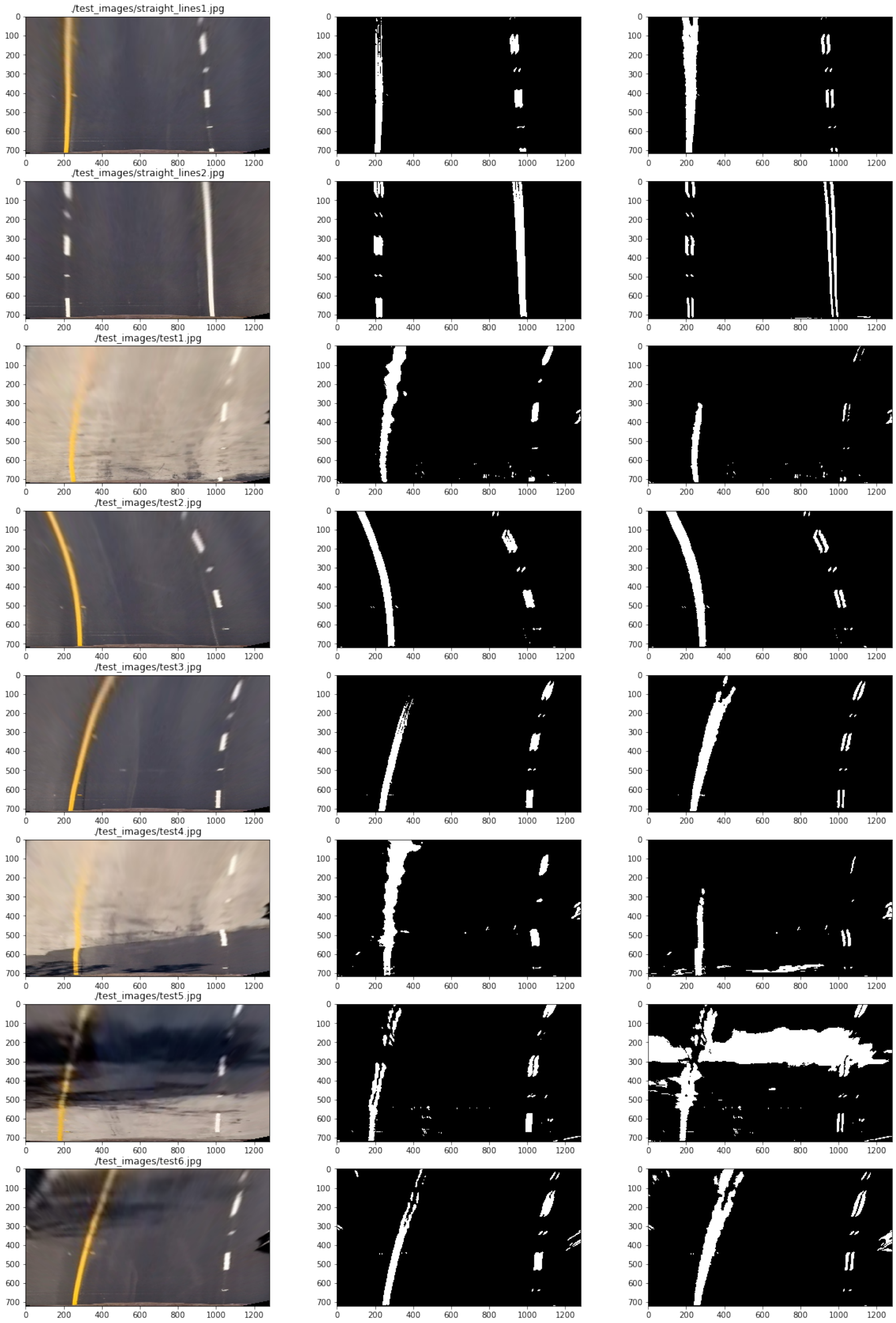


In [14]:

```
def rgb_select(image, ch=0, thresh=(0, 255)):
    channel = image[:,:,ch]
    binary = np.zeros_like(channel)
    binary[(channel > thresh[0]) & (channel <= thresh[1])] = 1
    return binary

def hls_select(image, ch=2, thresh=(0, 255)):
    hls = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
    channel = hls[:,:,ch]
    binary = np.zeros_like(channel)
    binary[(channel > thresh[0]) & (channel <= thresh[1])] = 1
    return binary

def hsv_select(image, ch=1, thresh=(0, 255)):
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    channel = hsv[:,:,ch]
    binary = np.zeros_like(channel)
    binary[(channel > thresh[0]) & (channel <= thresh[1])] = 1
    return binary

def pipeline(image):
    # Choose a Sobel kernel size
    ksize = 31 # Choose a larger odd number to smooth gradient measurement

    # Apply each of the thresholding functions
    gradx = abs_sobel_thresh(image, orient='x', sobel_kernel=ksize, thresh
    hls_s_binary = hls_select(image, ch=2, thresh=(100, 255))
    hsv_s_binary = hsv_select(image, ch=1, thresh=(100, 255))
    hls_v_binary = hls_select(image, ch=1, thresh=(120, 160))
    hls_h_binary = hls_select(image, ch=1, thresh=(15, 100))
    rgb_r_binary = rgb_select(image, ch=0, thresh=(220, 255))

    yellow_lines_binary, yellow_mask = yellow_lines(image)
    white_lines_binary, white_mask = white_lines_from_RGB(image)
    lane_mask = cv2.bitwise_or(yellow_mask, white_mask)

    combined1 = np.zeros_like(gradx)
    combined2 = np.zeros_like(gradx)

    combined1[((gradx == 1) | (rgb_r_binary == 1) | ((hls_s_binary == 1) &
    combined2[((gradx == 1) | (hsv_s_binary == 1))] = 1

    return combined1, combined2
```

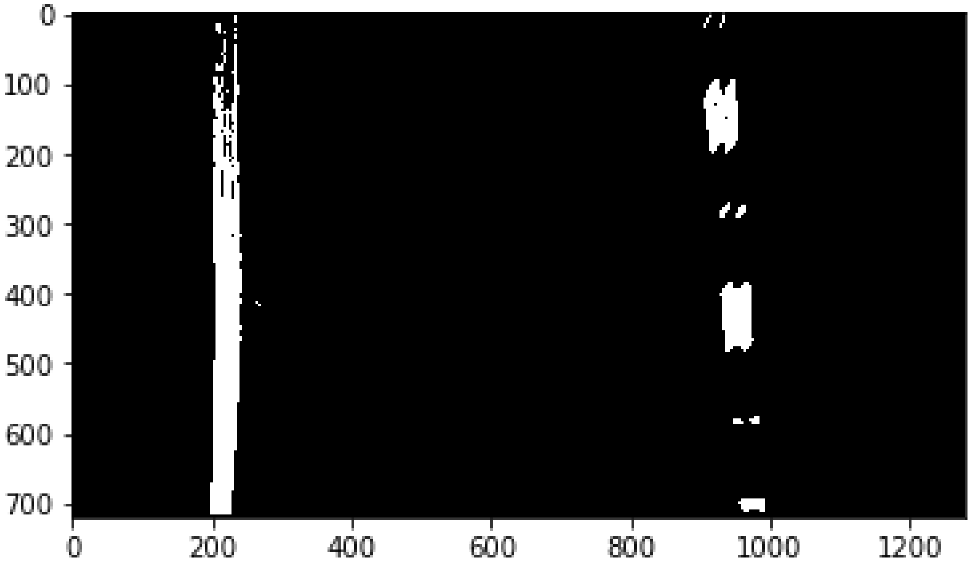
In [15]:

```
# try against all test images
images = glob.glob('./test_images/*.jpg')

# plots
rows = len(images)
cols = 3
fig = plt.figure(figsize=(20, 30))

for idx, fname in enumerate(images):
    img = cv2.imread(fname)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    undist = undistort_image(img, mtx, dist)
    unwarp,M,Minv = unwarp_image(undist, src, dst)
    # process through pipeline
    img_out1, img_out2 = pipeline(unwarp)

    plt.subplot(rows, cols, idx * cols + 1)
    plt.imshow(unwarp)
    plt.title(fname)
    plt.subplot(rows, cols, idx * cols + 2)
    plt.imshow(img_out1, cmap='gray')
    plt.subplot(rows, cols, idx * cols + 3)
    plt.imshow(img_out2, cmap='gray')
```

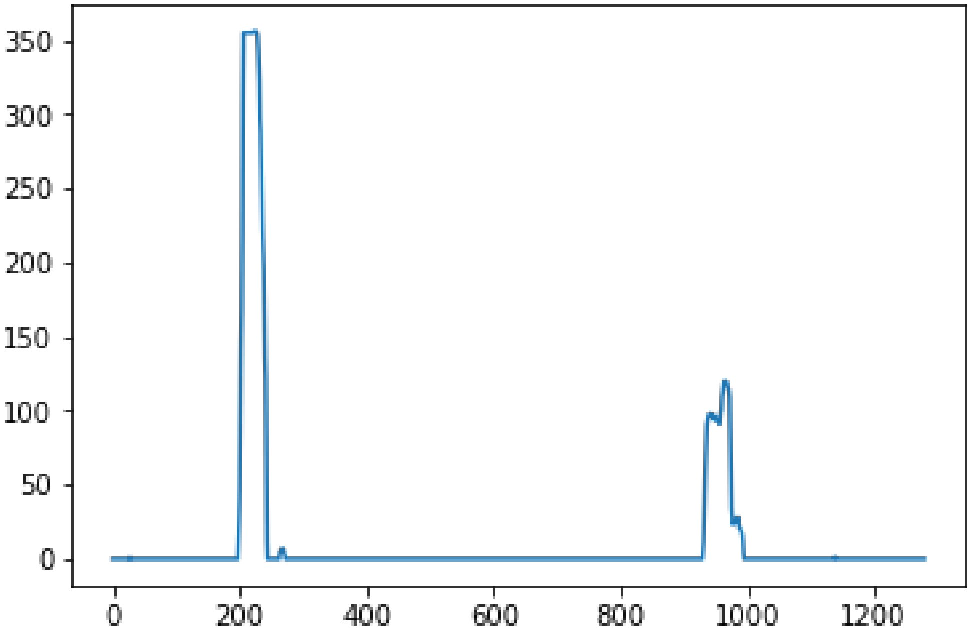

./test_images/straight_lines1.jpg
./test_images/straight_lines2.jpg
./test_images/test1.jpg
./test_images/test2.jpg
./test_images/test3.jpg
./test_images/test4.jpg
./test_images/test5.jpg
./test_images/test6.jpg
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

1.6 Locate the Lane Lines and Fit a Polynomial

```
In [16]: # ger outimage from pipeline
         img_out1, img_out2 = pipeline(unwarp_img)
         # display
         plt.imshow(img_out1, cmap="gray")
         plt.show()
```



```
In [17]: # Assuming you have created a warped binary image called "binary_warped"
         binary_warped = img_out1

         # Take a histogram of the bottom half of the image
         histogram = np.sum(binary_warped[int(binary_warped.shape[0]/2):,:], axis=
         plt.plot(histogram)
         plt.show()
```



```
In [18]: # Create an output image to draw on and  visualize the result
         out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
         # Find the peak of the left and right halves 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[:midpoint])
         rightx_base = np.argmax(histogram[midpoint:]) + midpoint

         print(leftx_base)
         print(rightx_base)
```

```
222
964
```

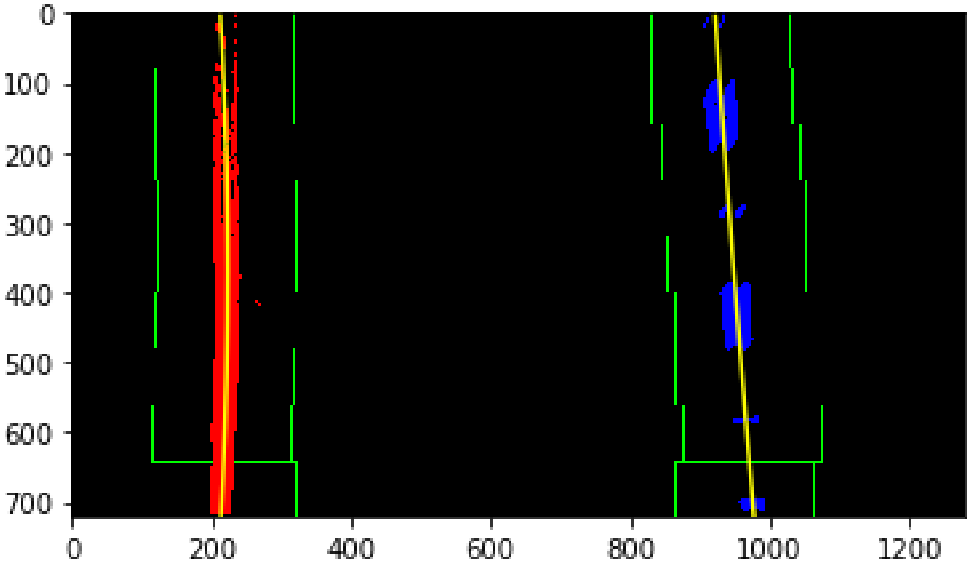
Sliding Window approach

```
In [19]: # Choose the number of sliding windows
         nwindows = 9
         # Set height of windows
         window_height = np.int(binary_warped.shape[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 each window
```

```
leftx_current = leftx_base
rightx_current = rightx_base
# Set the width of the windows +/- margin
margin = 100
# Set minimum number of pixels found to recenter window
minpix = 50
# Create empty lists to receive left and 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 and y (and right and left)
    win_y_low = binary_warped.shape[0] - (window+1)*window_height
    win_y_high = binary_warped.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_
    cv2.rectangle(out_img,(win_xright_low,win_y_low),(win_xright_high,win_
    # Identify the nonzero pixels in x and y within the window
    good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
    good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
    # 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 pos
    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]

# 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.shape[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 + right_fit[1]*ploty + right_fit[2]

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

```
Out[19]: (720, 0)
```



1.6.1 Left and Right Curvature (in pixels)¶

```
In [20]: y_eval = np.max(lefty)
         left_curverad = ((1 + (2*left_fit[0]*y_eval + left_fit[1])**2)**1.5) / np.
         right_curverad = ((1 + (2*right_fit[0]*y_eval + right_fit[1])**2)**1.5) /
         print("Left curvature : {:.2f} px  Right curvature : {:.2f} px".format(lef
         # Example values: 1926.74 1908.48
```

```
Left curvature : 6465.08 px  Right curvature : 35804.13 px
```

1.6.2 Left and Right Curvature (in meters)

```
In [21]: # Define conversions in x and y from pixels space to meters
         ym_per_pix = 30/720 # meters per pixel in y dimension
         xm_per_pix = 3.7/800 # meters per pixel in x dimension

         # Fit new polynomials to x,y in world space
         left_fit_cr = np.polyfit(lefty * ym_per_pix, leftx * xm_per_pix, 2)
         right_fit_cr = np.polyfit(righty * ym_per_pix, rightx * xm_per_pix, 2)

         # Calculate the new radii of curvature
         left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1]
         right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_c

         # Now our radius of curvature is in meters
         print("Left curvature : {:.2f} m  Right curvature : {:.2f} m".format(left_
         # Example values: 632.1 m    626.2 m
```

```
Left curvature : 2416.41 m  Right curvature : 13288.25 m
```

1.6.3 Distance of the car from the center of the lane

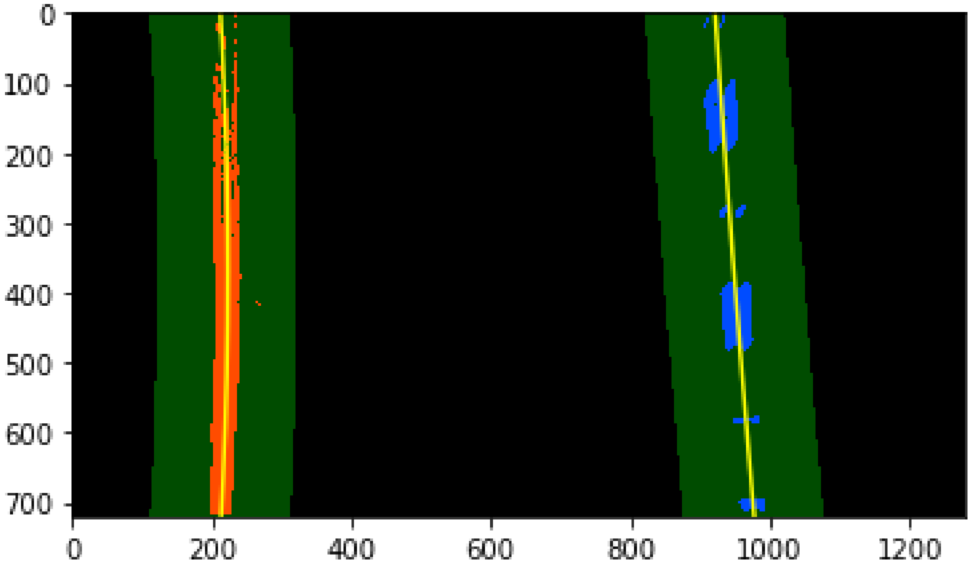
```
In [22]: img_center = binary_warped.shape[1] / 2
         lane_center = (left_fit[-1] + right_fit[-1]) / 2
         diff = lane_center - img_center
         print("Distance from center : {:.2f} m".format(diff * xm_per_pix))

         center_fit = (left_fit + right_fit) / 2
         center_curverad = ((1 + (2 * center_fit[0] * y_eval * ym_per_pix + center_
         print("Curvature : {:.2f} m ".format(center_curverad))
```

```
Distance from center : -0.34 m
Curvature : 15822.10 m
```

Plot Selection Window

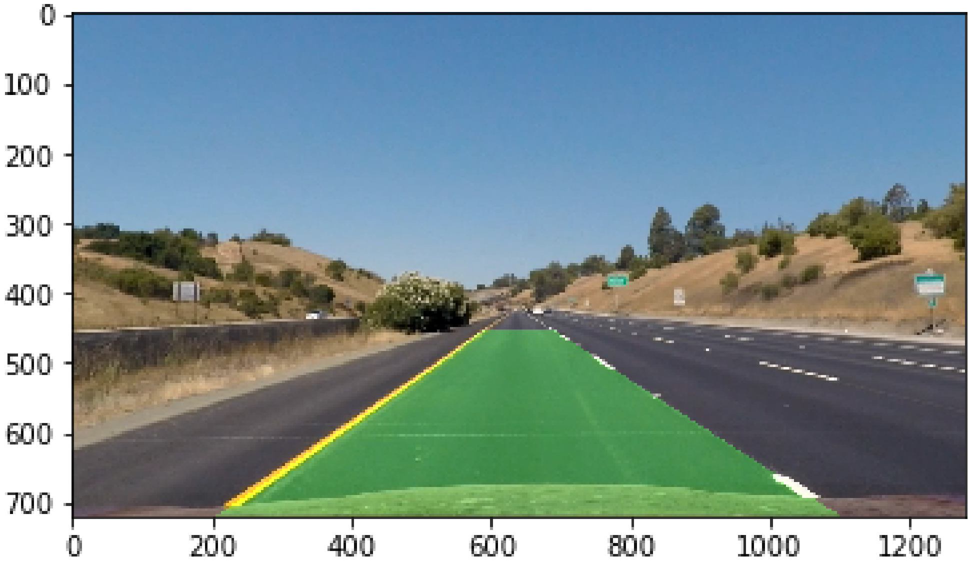
```
In [23]: # Create an image to draw on and an image to show the selection window
         out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
         window_img = np.zeros_like(out_img)
         # Color in left and right line pixels
         out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [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, pl
         left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fitx+
         left_line_pts = np.hstack((left_line_window1, left_line_window2))
         right_line_window1 = np.array([np.transpose(np.vstack([right_fitx-margin,
         right_line_window2 = np.array([np.flipud(np.transpose(np.vstack([right_fit
         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)
```

```
Out[23]: (720, 0)
```



Overlay

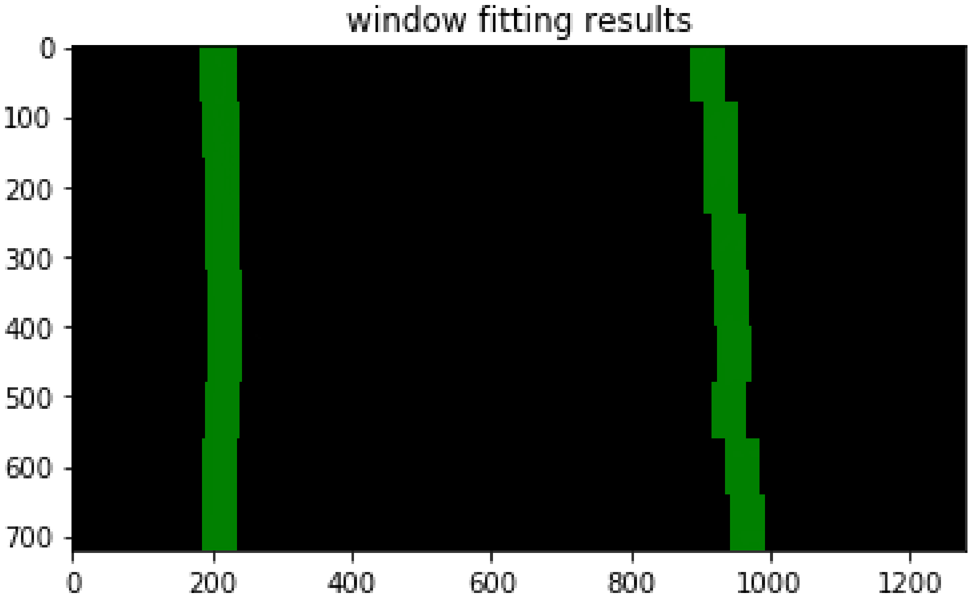
```
In [24]: warped = img_out1

         # Create an image to draw the lines on
         warp_zero = np.zeros_like(warped).astype(np.uint8)
         color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

         # Recast the x and y points into usable format for cv2.fillPoly()
         pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
         pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty]
         pts = np.hstack((pts_left, pts_right))

         # Draw the lane onto the warped blank image
         cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))

         # Warp the blank back to original image space using inverse perspective ma
         newwarp = cv2.warpPerspective(color_warp, Minv, (undist_img.shape[1], undi
         # Combine the result with the original image
         result = cv2.addWeighted(undist_img, 1, newwarp, 0.3, 0)
         plt.imshow(result)

Out[24]: <matplotlib.image.AxesImage at 0x116b4c390>
```



1.6.4 Alternate approach to Sliding Window for finding lane lines

```
In [25]: warped = img_out1

         # window settings
         window_width = 50
         window_height = 80 # Break image into 9 vertical layers since image height
         margin = 100 # How much to slide left and right for searching

         def window_mask(width, height, img_ref, center,level):
             output = np.zeros_like(img_ref)
             output[int(img_ref.shape[0]-(level+1)*height):int(img_ref.shape[0]-lev
             return output

         def find_window_centroids(image, window_width, window_height, margin):
```

```
    window_centroids = [] # Store the (left,right) window centroid positio
    window = np.ones(window_width) # Create our window template that we wi

    # First find the two starting positions for the left and right lane by
    # and then np.convolve the vertical image slice with the window templa

    # Sum quarter bottom of image to get slice, could use a different rati
    l_sum = np.sum(warped[int(3*warped.shape[0]/4):,:int(warped.shape[1]/2
    l_center = np.argmax(np.convolve(window,l_sum))-window_width/2
    r_sum = np.sum(warped[int(3*warped.shape[0]/4):,int(warped.shape[1]/2)
    r_center = np.argmax(np.convolve(window,r_sum))-window_width/2+int(war

    # Add what we found for the first layer
    window_centroids.append((l_center,r_center))

    # Go through each layer looking for max pixel locations
    for level in range(1,(int)(warped.shape[0]/window_height)):
        # convolve the window into the vertical slice of the image
        image_layer = np.sum(warped[int(warped.shape[0]-(level+1)*window_h
        conv_signal = np.convolve(window, image_layer)
        # Find the best left centroid by using past left center as a refer
        # Use window_width/2 as offset because convolution signal referenc
        offset = window_width/2
        l_min_index = int(max(l_center+offset-margin,0))
        l_max_index = int(min(l_center+offset+margin,warped.shape[1]))
        l_center = np.argmax(conv_signal[l_min_index:l_max_index])+l_min_i
        # Find the best right centroid by using past right center as a re
        r_min_index = int(max(r_center+offset-margin,0))
        r_max_index = int(min(r_center+offset+margin,warped.shape[1]))
        r_center = np.argmax(conv_signal[r_min_index:r_max_index])+r_min_i
        # Add what we found for that layer
        window_centroids.append((l_center,r_center))

    return window_centroids

window_centroids = find_window_centroids(warped, window_width, window_heig

# If we found any window centers
if len(window_centroids) > 0:

    # Points used to draw all the left and right windows
    l_points = np.zeros_like(warped)
    r_points = np.zeros_like(warped)

    # Go through each level and draw the windows
    for level in range(0,len(window_centroids)):
        # Window_mask is a function to draw window areas
        l_mask = window_mask(window_width,window_height,warped,window_cent
        r_mask = window_mask(window_width,window_height,warped,window_cent
        # Add graphic points from window mask here to total pixels found
        l_points[(l_points == 255) | ((l_mask == 1) ) ] = 255
        r_points[(r_points == 255) | ((r_mask == 1) ) ] = 255

    # Draw the results
    template = np.array(r_points+l_points,np.uint8) # add both left and ri
    zero_channel = np.zeros_like(template) # create a zero color channle
    template = np.array(cv2.merge((zero_channel,template,zero_channel)),np
    warpage = np.array(cv2.merge((warped,warped,warped)),np.uint8) # makin
    output = cv2.addWeighted(warpage, 1, template, 0.5, 0.0) # overlay the

# If no window centers found, just display orginal road image
else:
    output = np.array(cv2.merge((warped,warped,warped)),np.uint8)

# Display the final results
plt.imshow(output)
plt.title('window fitting results')
plt.show()
```


window fitting results