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.

Camera Calibration

1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The steps involved in calibrating the camera is as follows :

Let `objp` represent the undistorted flat 3D grid points of `m x n` dimensional corners which is assumed to be at z=0. Hence it will always be of the type (x,y,0). The corners of the chessboard can be found using the function `cv2.findChessboardCorners(gray, (9,6),None)` which returns the 2D pixel co-ordinates of each corner in the image.

Now let's initialize two lists viz. `objpoints` and `imgpoints`. Every time we detect all the corners in a chessboard image, `objpoints` appends a copy of `objp` and `imgpoints` appends a copy of the corner co-ordinates.

To obtain the camera matrix and distortion co-efficients, we use the `cv2.calibrateCamera()` function and pass the lists `imgpoints` and `objpoints` and the shape of the image as arguments. The function returns the distortion coefficients `dist` and camera matrix `mtx`.

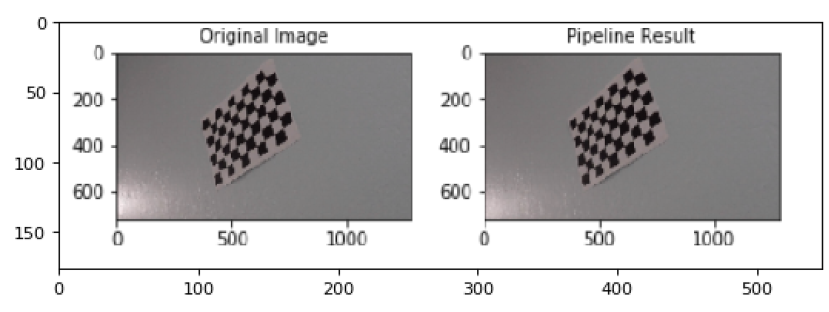
Now we use the `cv2.undistort()` function with the image, camera matrix and distortion co-efficients as arguments to undistort any image

alt text

In [9]:
```
import matplotlib.pyplot as plt
import cv2
from matplotlib.pyplot import figure

figure(figsize=(8, 6), dpi=80)
plt.imshow(cv2.imread("output_images/calibration.png"))
```

Out[9]: `<matplotlib.image.AxesImage at 0x7fc4e5734e48>`



Pipeline (single images)

1. Provide an example of a distortion-corrected image.

To demonstrate distortion-correction, we utilize the camera matrix and distortion coefficients we obtained in the previous step and pass them along with the image to `cv2.undistort()` function. The output looks like this :

In [12]:
```
figure(figsize=(8, 6), dpi=80)
plt.imshow(cv2.cvtColor(cv2.imread("output_images/undistorted.png"),cv2.COLOR_BGR2RGB))
```

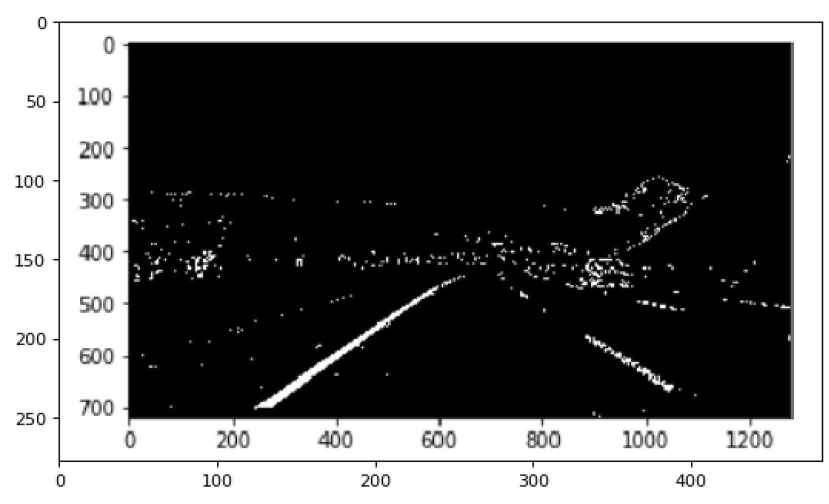
Out[12]: `<matplotlib.image.AxesImage at 0x7fc4bd898550>`

2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate the binary image. First I converted the image to HLS colourspace and set thresholds for the `Saturation` channel. Similarly I did a convolution on the image with `Sobel_x` filter and set thresholds on the gradient. Like this, I created the binary image which gives the following result

In [13]:
```
figure(figsize=(8, 6), dpi=80)
plt.imshow(cv2.cvtColor(cv2.imread("output_images/thresh.png"),cv2.COLOR_BGR2RGB))
```

Out[13]: `<matplotlib.image.AxesImage at 0x7fc4ac5ff9e8>`



3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

```
# Find the perspective transform of the image
def Perspective(self,img):
    y=img.shape[0]
    x=img.shape[1]
    offset=300
    src = np.float32([[190, 720], [596, 447],[685, 447],[1125, 720]])
    dst = np.float32([[offset, y],[offset, 0],[x-offset, 0],[x-offset, y]])
    M = cv2.getPerspectiveTransform(src, dst)
    M_inv = cv2.getPerspectiveTransform(dst,src)
    return_image = cv2.warpPerspective(img, M, (x,y), flags=cv2.INTER_LINEAR)
    return return_image,M_inv
```

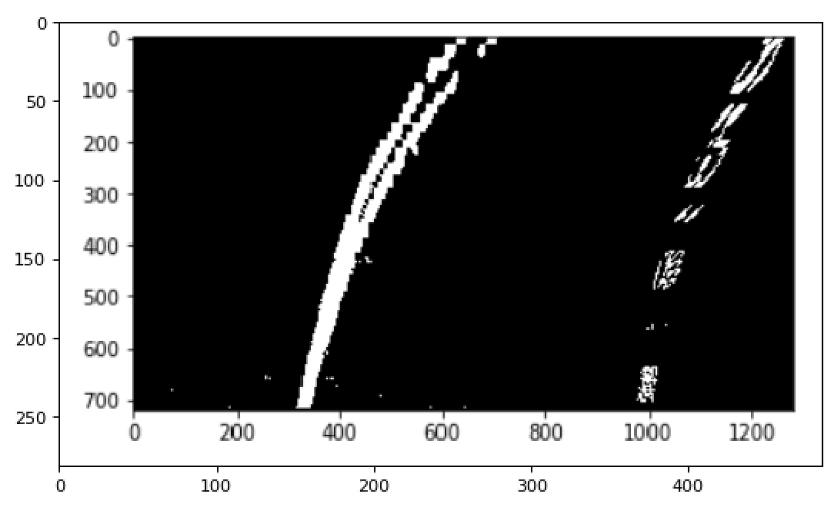
The perspective transform was achieved by defining source and destination points and passing them to the `cv2.getPerspectiveTransform()` function in the `LaneLineDetector()` class. The four source points were found by trial and error method and the corresponding destination points were defined by using a offset variable so that its uniform from both sides.

Once the perspective transform was calculated, we use the `cv2.warpPerspective()` with the image nd transform as inputs to warp the image.

The Perspective function is contained inside the LaneLines() class.

In [15]:
```
figure(figsize=(8, 6), dpi=80)
plt.imshow(cv2.cvtColor(cv2.imread("output_images/Perspective_tf.png"),cv2.COLOR_BGR2RGB))
```

Out[15]: `<matplotlib.image.AxesImage at 0x7fc4ac53c588>`



4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

I used the `find_lane_pixels()` and `fit_polynomial()` functions in the `LaneLineDetector()` class to identify the lane lines and fit their positions with a polynomial.

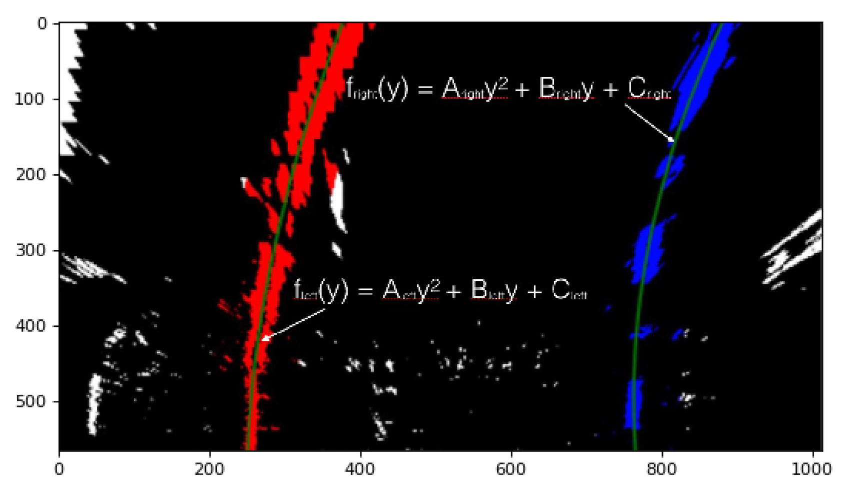
`find_lane_pixels()` : This function uses a histogram based method to extract left and right line pixel positions

- Take a histogram of the bottom half of the image and find the peaks in the left and right parts of the image. This helps us find the starting points for the left and right lane lines.
- The hyperparameters like number of sliding windows, width of the windows and minimum number of pixels which helps to recenter the window are set after this.
- For each window we Identify the nonzero pixels in x and y within the window and recenter the window to the mean of the previous window's non-zero pixels
- Then after all the window steps, we extract the x and y location of the total selected pixels and it is passed onto the `fit_polynomial()` function

`find_lane_pixels()` : This function takes in the x and y locations of the selected pixels from the `find_lane_pixels()` and fits a second order polynomial of the formula Ax^2 + Bx + C to it.

In [16]:
```
figure(figsize=(8, 6), dpi=80)
plt.imshow(cv2.cvtColor(cv2.imread("examples/color_fit_lines.jpg"),cv2.COLOR_BGR2RGB))
```

Out[16]: `<matplotlib.image.AxesImage at 0x7fc4ac513a90>`



5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

In the `measure_curvature_meters()` function of `LaneLineDetector()` class, we take the left and right x values of the lane lines along with the corresponding y values. The equation of the lane lines in the real world is obtained by using the `np.polyfit()` function on the inputs already multiplied with the conversion factors. Now that we have the coefficients of the quadratic equation, we can substitute this in the mathematical formulae for radius of curvature to obtain the radius.

$R = \frac{(1+(2Ay+B)^2)^{\frac{3}{2}}}{abs(2A)}$ - Formula for radius of curvature.

Position of car from center:

To find the position of car from center of the lane we need 2 things:

1. The car position- which is the center of the Image's horizontal dimension
2. Lane center- which can be found by taking the average of the x coordinates of the left and right lane lines. Once we get these two values we take there difference to get the deviation of the car from the center of the lane.

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

The `show_lane()` function of the `LaneLineDetector` class is used to plot the result back onto the road. The result obtained is as follows :

In [17]:
```
figure(figsize=(8, 6), dpi=80)
plt.imshow(cv2.cvtColor(cv2.imread("output_images/image_working.png"),cv2.COLOR_BGR2RGB))
```

Out[17]: `<matplotlib.image.AxesImage at 0x7fc4ac476550>`

Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

Here's a [link to my video result]

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

Some of the challenges I found while doing this project were plotted lane jumping around due to issues in thresholding, then finding the vehicle position and offset. However these problems were resolved to some extend. The pipeline will likely fail when we go in underpasses of bridges etc where suddenly the light condition changes. This can be overcome by fine-tuning the color threshold values.

In [ ]:

In [ ]: