**Writeup Template**

**You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.**

マークアップファイルとして提出したい場合は、このファイルをあなたの書き込みのテンプレートとして使用できますが、他の方法を使用して自由に**pdf**を提出することもできます。

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.
* Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

このプロジェクトの目標/手順は次のとおりです。

①1組のチェスボード画像を与えられたカメラの較正行列と歪み係数を計算する。

②生の画像に歪み補正を適用します。

③色変換、グラデーションなどを使用して、しきい値付きバイナリイメージを作成します。

④パースペクティブ変換を適用してバイナリイメージ（「鳥瞰図」）を修正します。

⑤レーンピクセルを検出し、レーン境界を見つけるために適合する。

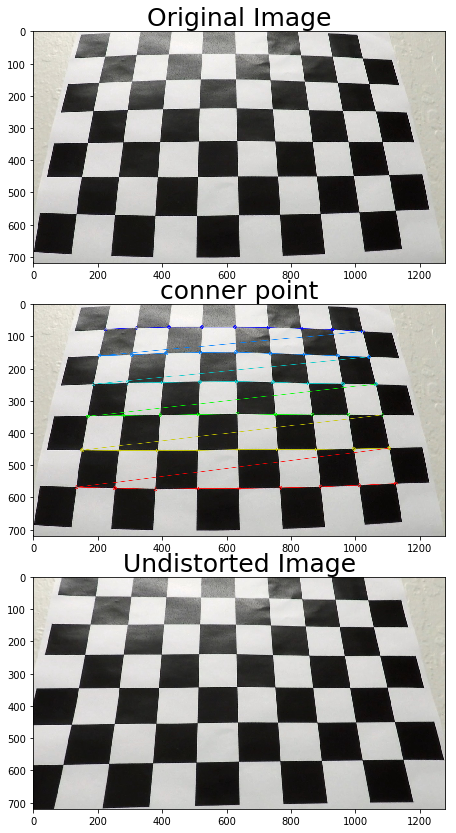
⑥中心に対する車線の曲率および車両の位置を決定する。

⑦検出された車線境界を元の画像に戻します。

⑧車線境界の視覚的な表示と、車線の曲率および車両位置の数値推定を出力します。

**[Rubric](https://review.udacity.com/" \l "!/rubrics/571/view) Points**

**Here I will consider the rubric points individually and describe how I addressed each point in my implementation.**

**Camera Calibration**

私はまず９ｘ６点のコーナを持つチェッカーボード上を**Z=0**平面、描かれた各コーナーを**X**、**Y**座標とする実座標系を定義しました。

そして画面上に表示座標系を設定し、撮影した先のチェッカーボードの各コーナー９ｘ６点のピクセルを表示座標として記録しました。

次に向きや位置を異なる２０通りパターンで移動させて、各パターンにおけるコーナー９ｘ６点の表示位置を記録しました。

各パターン、各コーナーの**XYZ**値はリスト**objpoints**に格納し、表示ピクセル値はリスト**imgpoints**に格納しました。

私は**objpoints**と**imgpoints**を入力とするcv2.calibrateCamera()関数を使用してカメラの較正および歪み係数を計算しました。この歪み補正をcv2.undistort()関数を使ってテスト画像に適用し、右図の結果を得ました：

**Pipeline (single images)**

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



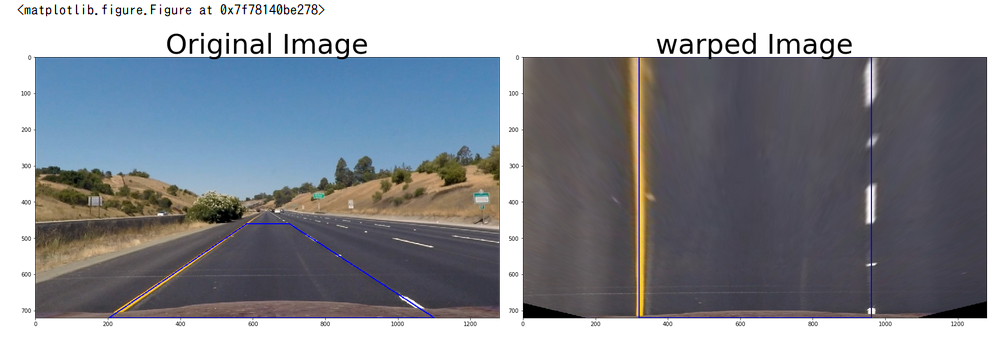
**2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.**

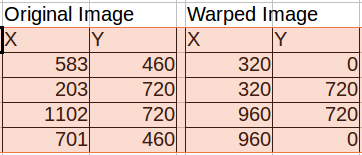
**3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.**

私はまず、直進中の撮影画像（左下）をcv2.undistort()関数で補正して基準画像を作成した。

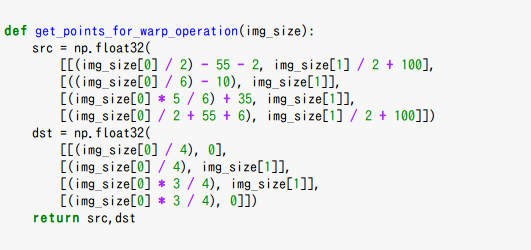
次に直進中の基準画像から鳥瞰画像への変換キーとなる台形エリアのコーナ４点を検討した。

それらが示すエリアが、鳥瞰図上の横位置1/4〜3/4エリアに表示されるような変換基準点の抽出を試みた。



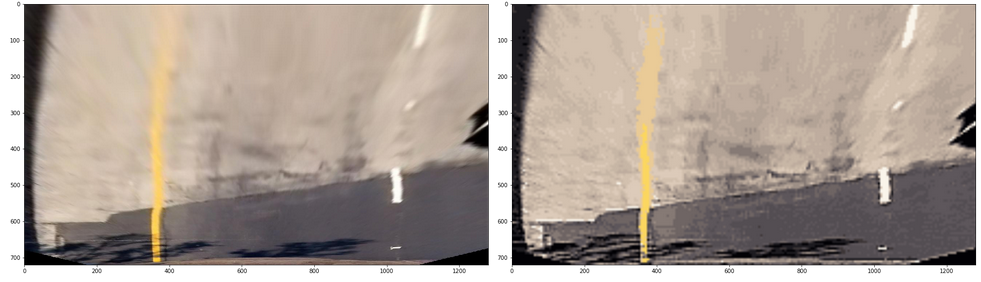
一連の作業について得た対応点を、下記関数として記録し再利用可能とした。

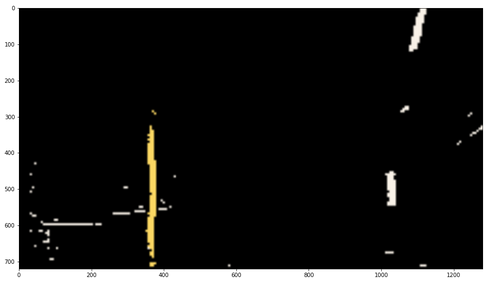
４つの対応点より鳥瞰画像変換のためのM行列をcv2.getPerspectiveTransform関数を用いて取得した。  
また、得たM行列と cv2.warpPerspective関数を用いて鳥瞰画像を作成できた。



次に鳥瞰図より、画像の中に含まれるレーンの抽出を行う。レーンは鳥瞰図の中に含まれる情報のうちユニークな色を持ち、且つ縦方向に連なる特徴があることに着目する。

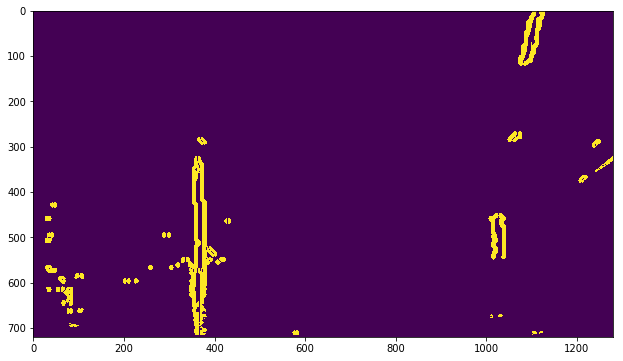
まず鳥瞰図に対してcv2.kmeans関数を用いて含まれるRGB情報を１６パターンに減色する処理を加えた。

次に、１６パターンの中から面積の占める割合が多いラベル、横方向に長いラベルを除外することで、ユニークな色を持ち、且つ縦方向に連なる特徴を持つラベルをレーンとして抽出した。



残った僅かな横方向のノイズ分を除去するためにX方向に対するSOBELを適用した。

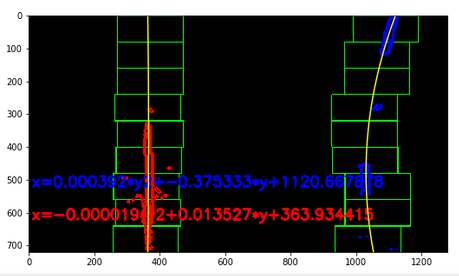
最後に残った画素を２値化して、レーン検出に必要なバイナリイメージを作成した。



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

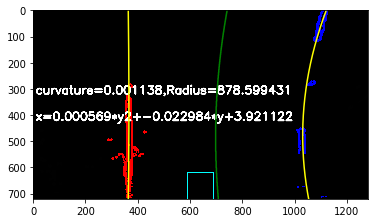
４−１　左右レーンを示す近似曲線

レーン検出に必要なバイナリイメージにスライディング法を使用して、ヒストグラムのピーク（レーンであろうと仮定する画素の塊）を基準に、「レーン抽出のための小エリアを縦方向」に連ねて、含まれる画素より表示レーンを２次方程式による近似曲線で示すこととした。

本サンプル画像の場合、結果右レーン（青）は式x=0.000392\*y^2+-0.375333\*y+1120.667878（単位ピクセル）、左レーン（赤）はx=-0.000019\*y^2+0.013527\*y+363.934415（単位ピクセル）の近似曲線として計算できた。

４−２　レーンを示す近似曲線（中央位置）

次に左右レーン毎に取得した２次方程式を用いて黄色線を描画した。その後左右の黄色線について、高さy位置に置ける横ｘ中心点を記録した。記録した中心点ｘ，ｙの集まりからレーン中央位置を示す２次方程式より得られた式x=0.000187\*y^2+-0.181181\*y+741.833852（単位ピクセル）で示す近似曲線（緑線）を「レーンを示す近似曲線」と定めた。



**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.**

５−１　車両のレーン逸脱距離

自車の先端はカメラ基準となるので画面下部中心四角枠位置（６４０，７２０）付近に位置すると仮定する。  
 緑線の下部ｘ位置はおよそ708ピクセルであり、その間のズレ１２ピクセルを逸脱距離と仮定する。

プロジェクト内の情報より、画像単位[ピクセル]と実空間単位[m]の変換率は以下のとおりである。

ym\_per\_pix = 30/720 # meters per pixel in y dimension  
xm\_per\_pix = 3.7/700 # meters per pixel in x dimension

車両とレーンのズレ１２ピクセルは実空間上で約0.0634mと推定することができる。

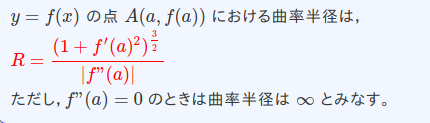
５−２　実際の道路曲率半径（勾配）

４−２　でレーン中央位置を示す近似曲線を求める際に記録した左右黄色線の中心点情報ｘ,ｙ情報にxm\_per\_pix とym\_per\_pix を掛けて実空間上での位置距離情報X,Yを計算した。

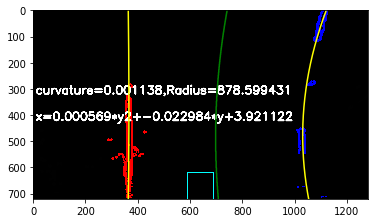
得たX,Y位置情報に対して２次方程式で示すこととした。

その結果、x=0.000569\*y^2+-0.022984\*y+3.921122　（単位ｍ）による近似曲線を得た。

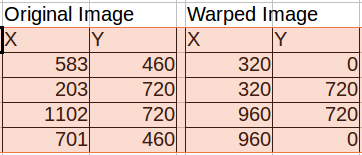
ところで道路曲率半径は以下式で定義できる。



x=f(ｙ） ＝ 0.000569\*y^2+-0.022984\*y+3.921122の道路で、自車位置y=720ピクセルなので  
ｆ’(a) ＝2.＊0.000569＊（720＊ym\_per\_pix）-0.022984  
ｆ’’(a) ＝ 2.＊0.000569　であるから、　道路曲率半径は878.599431ｍと計算できた。

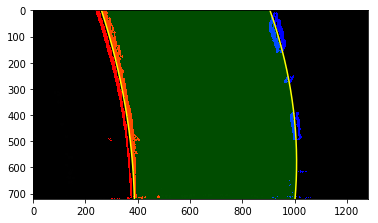


**6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.**

最後に私は鳥瞰図上で得た情報を元の基準画像に逆写像することに挑戦した。基準座標から鳥瞰図の変換で用いた4点の対応は右表のとおりであった。

そこでcv2.getPerspectiveTransform関数の引数に対して入力srcと出力dstを入れ替えた逆M行列を求めた。

鳥瞰図上から基準画像上に逆写像するために私は逆M行列とcv2.perspectiveTransform関数を使用した。  
各左右レーンのX,Y位置を鳥瞰図上で取得し、基準画像に変換したものが下図の結果である。





**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!).**

以上の処理を組み合わせて作成した動画ファイルを提出する。  
開発したレーン検出器で自分の進むべきレーンを指し示す様子を動画ファイル上で確認できる。



**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?**

・デッドレコニング機能の不足

左右レーン近似曲線の計算結果をそのまま利用したことで次の２つの課題が発生した。

①道路近似曲線（レーン中央）、道路曲率半径の値等が毎回ブレて安定しない。

②左右いずれかのレーンが正しく検出できなかった場合、精度や結果に大きく得依拠する。

これらの対応としては、過去に求めたレーンの２次式係数、近似曲線、スライディングウィンドウの基点・開始位置を長期間記録し、もしもの際に補完／バックアップできるようなデッドレコニング機能を追加する必要がある。

・画像改質の強化

kmeanによる減色処理、あるいはHLSやエッジなどについては、解像度の不足する遠方や日影の影響でコントラストの確保が用意ではない（下図の場合、鳥瞰図上部のレーンが道路と同化してしまった)。

さらなる画像の事前の改質作業が望まれる。

