Callibrating the Camera

Open cv2 provides an excellent function for this called as `cv2.calibrateCamera()`.

The code acan be located in output.inpyb in an fucntion called callibrate

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of coordinates, and `objpoints` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:

This is an example of undistorted image, which is done through the undistort function.

This is original image

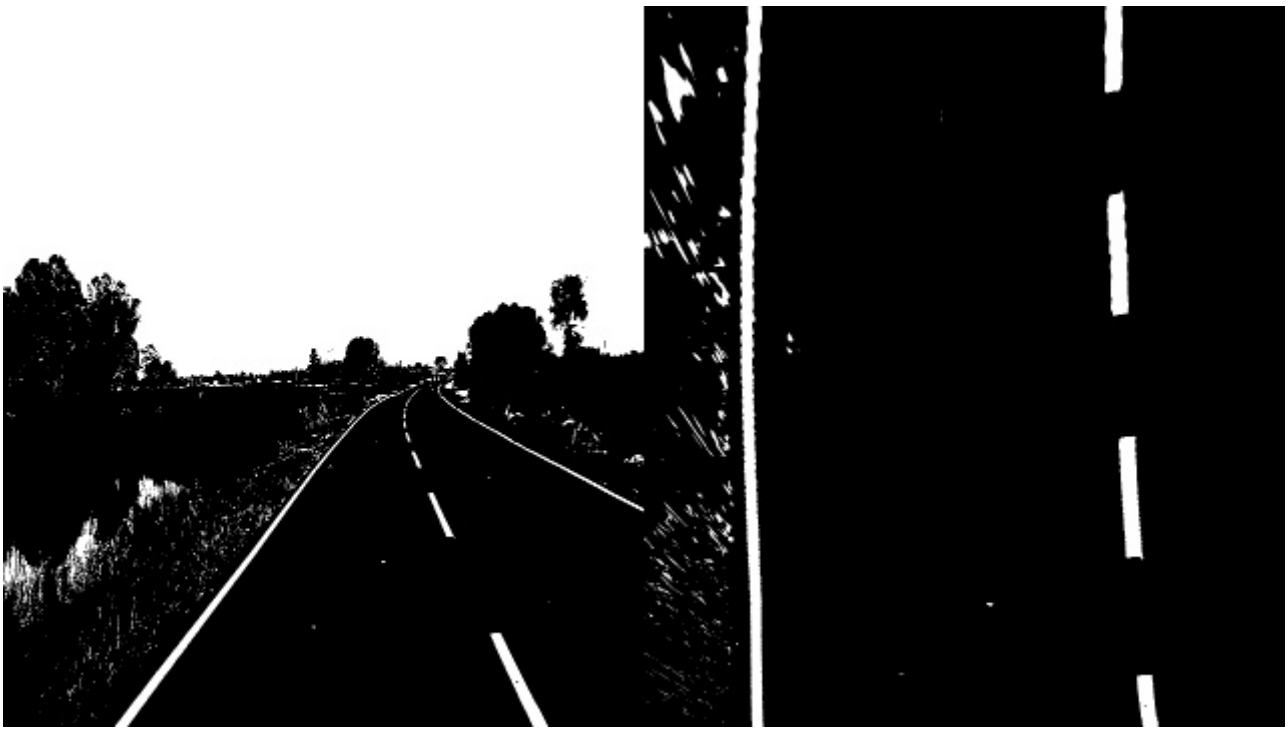
Image after removing distortion

You can see some part is removed from the lower rigth corner of the image after distortion is removed.

So now the thing is we need to detect lane lines and for that we can use various propoerties of them , like they are of white color have high Hue value, and they have edges.

I used a combination of color and gradient thresholds to generate a binary image (thresholding steps at lines # through # in `another_file.py`). Here's an example of my output for this step. (note: this is not actually from one of the test images) In this i have only used the HSV and RGB thresholding other technquies like sobel were not used as they were not giving an good result

- This is an example where at first lane is detected in image plane and then transformed inside birdeye view.



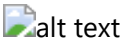
So now is converting the image in bird eye view.

For converting the image to bird eye view open cv has an extraordinary function know as `getPerspectiveTransform` this takes 4 points in a plane and convert it into another derired plane.As cleared by name it changes the perspectivetransform of the image.In our case the desired plane is plane parallel to the road. Now think of a drone is flying parallel to the car and taking pictures od road, wont it be same as picture shown in right side of the above picture??
Thankfully this magic is done by mathematics itself.

The `warp()` function takes as inputs an image (`img`). I chose the hardcode the source and destination points in the following manner:

Here is Hyperparameters for the same

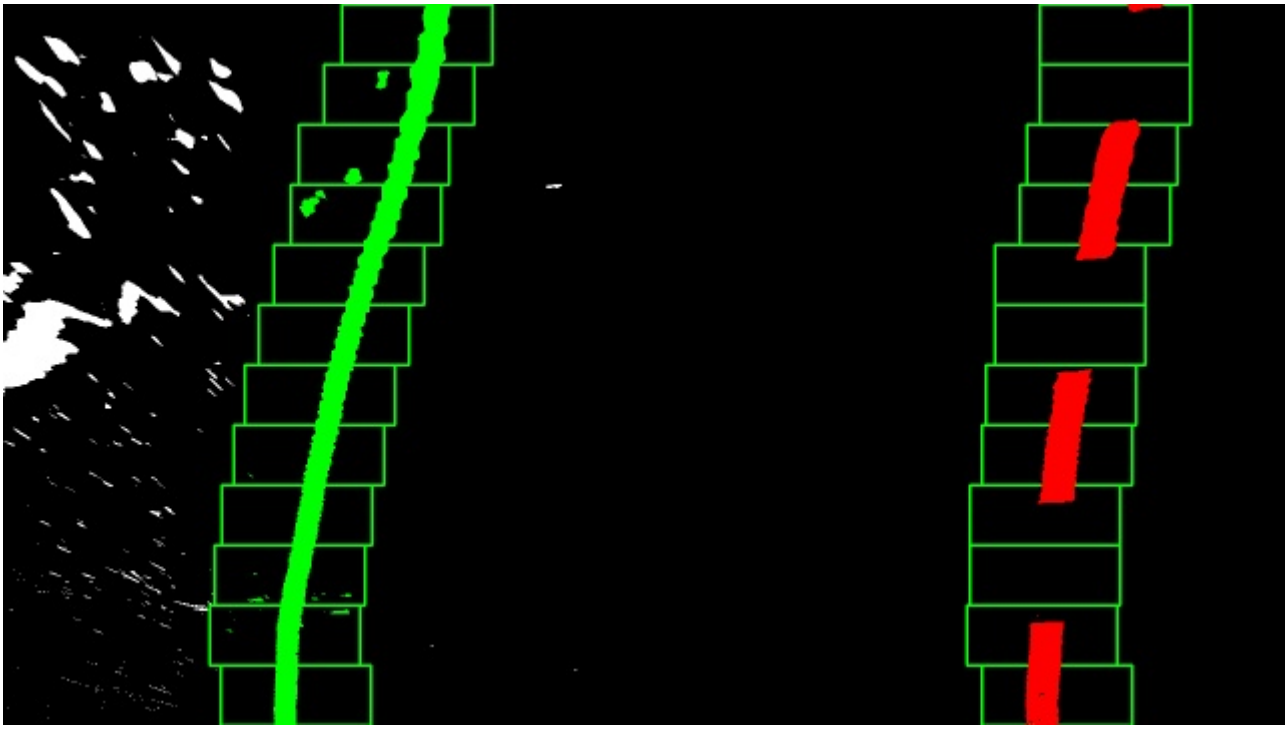
Source	Destination
295,720	150,720
1146,720	150, 0
850,430	1200, 0
675,425	1200, 720



I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image. This function returns an warp immage and 2 Matrix calles as M and Minv,whish are just transformation matrices from cource to sedtimnation and destination to source respectively.

So after converting it into bird-eye view next thing that comes is detecting lanes and taking useful information out of it.

- I have plotted an histogram on the image and to find the points and then using this i have found out the Polynomial and the positions



5.Now the bigger thing is finding raddius of curvature and the utilising it for our program.

Motivation for this is, considering the road width to be constant and then we have Hyperparameters is `warp_transformation` function above in the table, the destinations points are also constant there so for both lower part of the trapezium as well as the upper part of the trapezium, distance along x is 720 pixels. Now we have used this fact to plot an 2 degree curve along the Road lines. After knowing the equation of it, I have differentiated with respect to y

alt text

After finding raddius of curvature of both of them , I then found deflection from centre.

6. After doing all these things here is the output of result.

I implemented this in the same file with the finction named as `in the functiondraw_lines()`. Here is an example of my result on a test image:

Discussion of whole Pipeline

1.calibrated the camera
2.Undistort the image after points obtained from calibration
3.Mask the image using The function color_transform(output is an binary image)

4.Warp(bird eye view) the image(i.e Input: Undistorted image Outputs: are warped image,and 2 matrices)

```
Warp
Input->Undistorted Image
Output-> warped Image, and 2 transformation matrices
```

5.Create an warptransformation to the warped image with Minv matrics for giving it the same perspective

6.Apply color transformation to the warpedimage(using the function color_transformation)

7.Get an output image after using addWeight funcntion to the undistorted image and new warp image

Here is what i am talking in 5th and 7th step.

- 5th -> See lower part i.e after reverse trasnformation
- 7th -> See upper part of image, you can that common part of both the image upper and lower is somewhat bright in the upper image. That is what addWeight function does.

8.Fit polynomial using function fit_polynomial whose inputs is the image obtained in 6th step

9.Calculate the curavature using function known as measure_curvature_real

10.Draw lines on undistorted image these are the lines for lanes,This is output

11.put texts like curvature raddius and centre deflection to the given image

14.Display it using opencv

Discussion

Problem faced during this.

One Major this which i was facing is in finding the perfect coordinates for warptransformation, that i have solved using trackbar window techniques.

Future Plans

- To implement it along with cuda for better speed
- I have plans to implement Kalman Filters for tracking the lanes as well as developing another algorithm so that i can use both of them in a PID type function so that if one algo fails other will do its job.
- Another major thing is regrding Warp transformation, it is one the most important part in this whole function, using variable coordinates will fix the headache for failure of this pipeline during sharp turns.Currently i am working on an prototype for implementing an function which keep track of coordinate used in previous frame then calls warp_transformation function with those coordinates, it is somewhat like kalman filter.
- For pinpoint accuracy of raddius of curvature. Integration along with IMU can be done. Currently working on an different method for same.
- Algorithm to deal with Variable lightning conditions.