Advanced Lane Finding Project

The script of project is presented in file ***advanced_lane_findings.ipynb***

The 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.

Camera Calibration

For the camera calibration the input calibration images with the chess boards were investigated. Open CV methods to find the calibration matrix were used

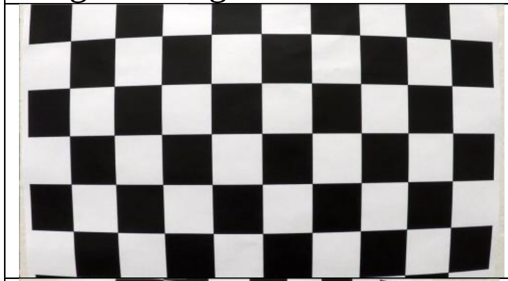
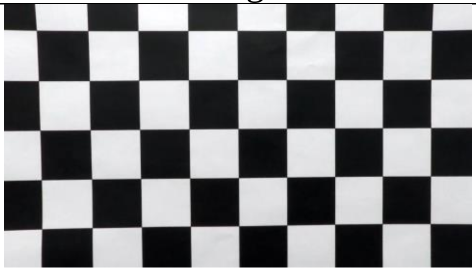
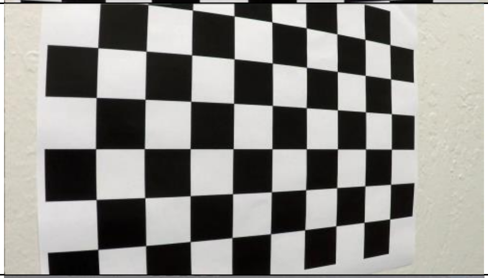
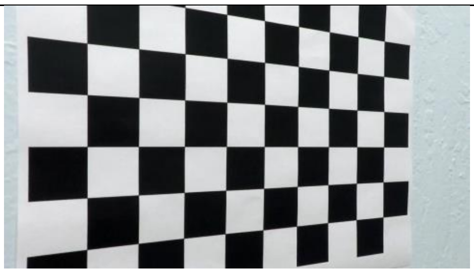
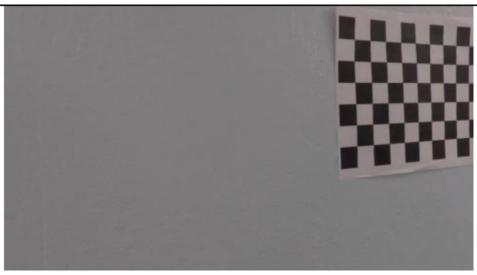
All the calibration functionality is provided in class ***Chess_calibration_points*** inside file ***"sources/calibration.py"***

Object of this class is initialized by set of input images with chess board, so calibration matrix is calculated on initialization.

For all the other images which is needed to be undistorted the method ***undist_image*** of current object is used.

The chess undistorted images are provided in the folder "***output_images/calibration"***

Some examples of calibration:

Original image	Undistorted image
	
	
	

Undistorted raw images are presented in folder ***“output_images/undist”***. Some examples:

Binary image creation

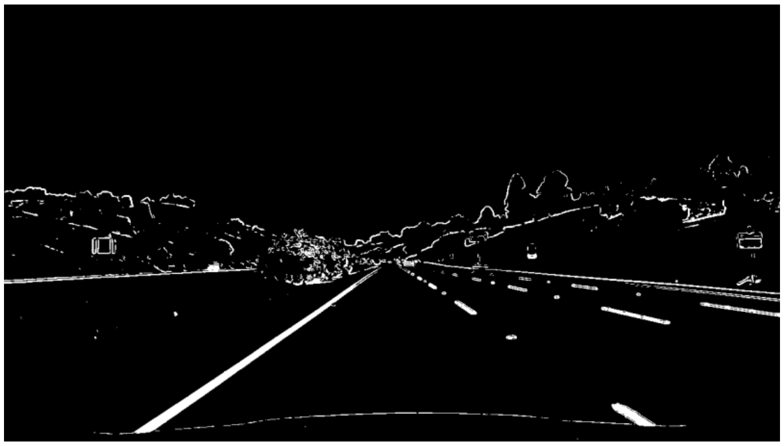
All the implemented binary transformations are provided in file ***"sources/ binary_outputs.py".***

Such a gradients are implemented there:

- ***get_sobel_binary*** – sobel x or y gradient
- ***get_hls_s_binary*** – HLS color space S channel gradient
- ***get_hls_l_binary*** - HLS color space L channel gradient
- ***get_yellow_binary*** – "yellow" and "white" colors gradient
- ***get_dir_binary*** – direction of the gradient
- ***get_mag_binary*** – magnitude of the gradient

The combination of the current gradients is presented in function ***get_binary*** which is located as separate function in file ***"sources/image_processing.py"***

The test output is presented in folder ***"output_images/binaries"***. Some examples of binary outputs:



Perspective transform

To provide perspective transformation the OpenCV function ***getPerspectiveTransform*** is used. Implementation is presented in file ***"sources/transformation.py".*** The class ***ImageWarper*** is presented here. Object of current class is initialized by *source* and *dest* polygons. All the images are transformed using method ***warp_image.*** Transformation is provided by matrix which where calculated using initialized polygons.

For the transformation such polygons were chosen (implementation is located inside *"sources/image_processing.py"* file):

```
def get_model_polygon (img):
    img_size = [img.shape[1], img.shape[0]]
    src = np.float32(
     [[(img_size[0] * 11/ 24), img_size[1] * 5/8],
     [((img_size[0] / 8) ), img_size[1]],
     [(img_size[0] *  11 / 12) , img_size[1]],
     [(img_size[0] * 13/ 24 ), img_size[1] * 5 / 8]])
    dst = np.float32(
     [[(img_size[0] / 6-50), 0],
     [(img_size[0] / 6-50), img_size[1]],
     [(img_size[0] * 5 / 6 -50), img_size[1]],
     [(img_size[0] * 5 / 6-50), 0]])
    return src, dst
```

For the model images, values of current polygons are:
Source polygon:

```
[[  586.66668701   450.        ]
 [  160.           720.        ]
 [ 1173.33337402   720.        ]
 [  693.33331299   450.        ]]
```

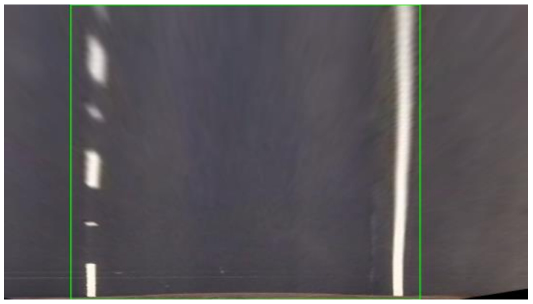
Dest polygon:

```
 [[  163.33332825     0.        ]
 [  163.33332825   720.        ]
 [ 1016.66668701   720.        ]
 [ 1016.66668701     0.        ]]
```

The transformation images are presented in folder "***output_images/bird_eye".***

Some examples of transformation:

Straight lines:



Curvature:

Binary images are presented in folder **“output_images/bird_eye_binaries”**

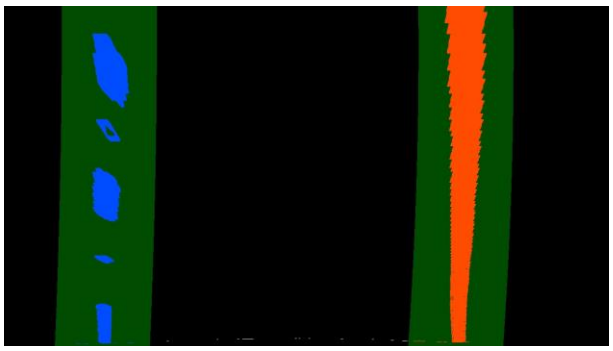
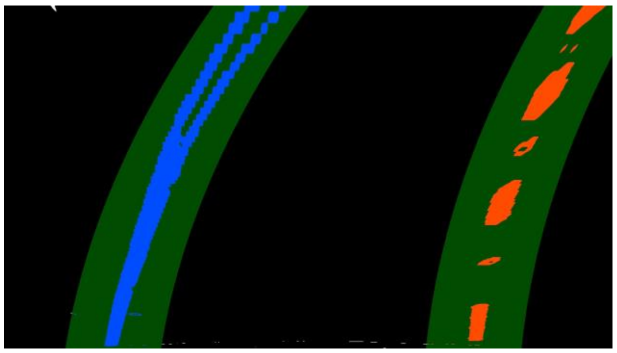
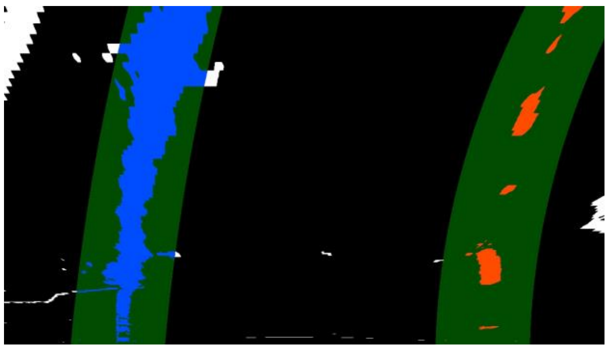
Lane-line pixels identification

For lane line pixels identification, the class ***LaneDetector*** (located in file *"sources/lane_detection.py")* is used. Current class contains left and right lane objects of class ***Line***.

To identify the lanes such methods are used:

- histogram calculation with sliding windows for the first images (method ***Line:: fit_polynomial***)
- search from prior polygons for the next images in videos (method ***Line::find_next_poly***)

The output results for line detection is presented in folder ***"output_images/lane_detection_bird_eye".*** Some of the results:

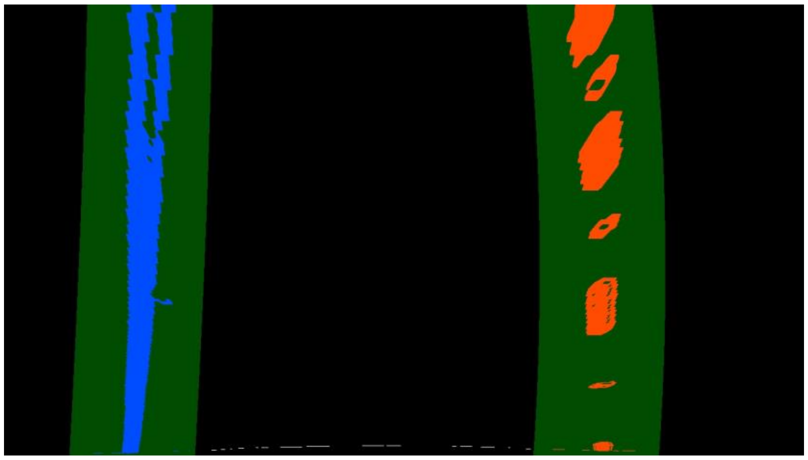
Radius Curvature and Car Offset

The radius of curvature is calculated using formula:

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

To provide radius of curvature in meters we need to identify meters for pixels value.

In case that lane width is usually ~3.7 meter and transformed image looks like:



I assume that current image has ~6 meters width and ~30 meters in perspective heights, so

```
ym_per_pix = ym_per_image/warped_image.shape[0] # meters per pixel in y dimension

xm_per_pix = xm_per_image/warped_image.shape[1] # meters per pixel in x dimension
```

where xm_per_image = 5 and ym_per_image = 30

The offset is calculated as difference between middle of the lanes and middle of the image (converted from pixels to meters)

Curvature calculation is presented in method **LaneDetector::get_curvature,** the offset in **LaneDetector:: get_offset.**

Here is the curvatures and offsets for the test images:

```
test6.jpg
Curvature  (9885.845315483477, 5952.7378399521695)
Center offset  -0.10312500000000001  meters
straight_lines2.jpg
Curvature  (5628.5874508696452, 3291.8705395992497)
Center offset  -0.24921875000000002  meters
test1.jpg
Curvature  (3716.08862140163, 451.17333008711148)
Center offset  -0.10742187500000001  meters
test2.jpg
Curvature  (389.94004840731407, 310.18485707542425)
Center offset  0.0171875  meters
straight_lines1.jpg
Curvature  (7789.0980269444108, 1580.0949818703452)
Center offset  -0.275  meters
test3.jpg
Curvature  (487.98428476188928, 543.42047261909431)
Center offset  -0.1375  meters
test5.jpg
Curvature  (654.5643285155345, 332.69874730810972)
Center offset  -0.24921875000000002  meters
test4.jpg
Curvature  (1663.3821351231024, 551.83400606323119)
Center offset  -0.038671875  meters
```

I expect that huge values of curvatures means that lanes are presented as straight lines there

Providing results back to the image

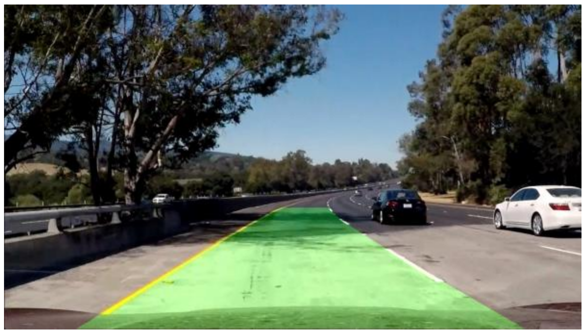
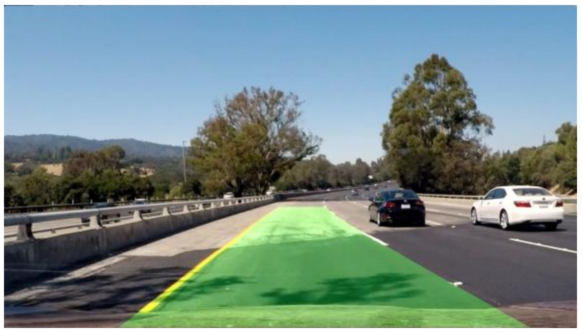
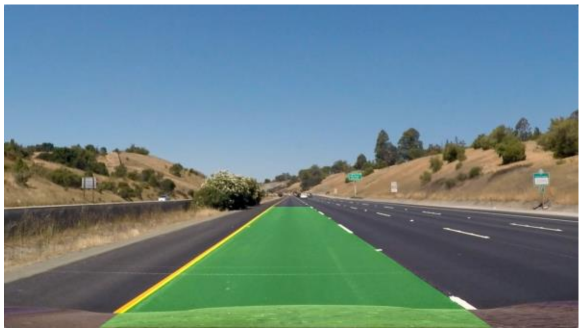
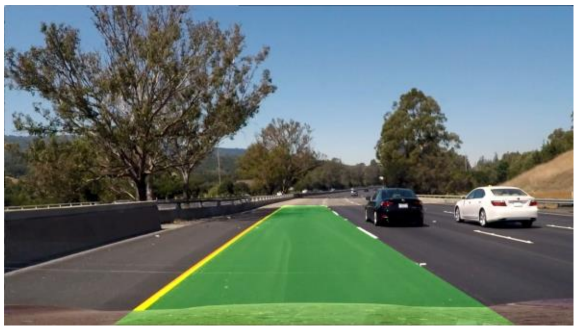
To provide the full image processing the class ***ImageProcessor*** was implemented (located in *"sources/image_processing.py")*. Current class processes the all needed image steps, such as:

- Undistort the image using *camera_calibrator* object
- Get binary image representation using *get_binary* function
- Wrap the image using *warper* with *source* and *dest* model polygons
- Identify the lanes using *LaneDetector*
- Provide lanes back to original image using *inverted_warper* with *dest* and *source* model polygons

To set the calculated lanes to the image the polygon is drawn using the method **LaneDetector:: get_the_polygon_image.** As mentioned above, current polygon is transformed to original image using the ***inverted_warper*** object.

The results for the test images are presented in folder ***"output_images/lane_detection"***

Some examples of output images:

Output videos

All the output videos are presented in **“output_videos”** folder. Unfortunately I’ve not won challenged videos, but I’ve tried 😊

Sanity checks

Road lanes on some frames cannot be correctly recognized because of road conditions, shades, etc. To avoid incorrect behavior, before lanes are accepted though the recognition, some sanity check is done. In case check is failed – lanes from previous accepted frame are used. The number of failed frames is regulated by **number_to_skip** value. If number of failed frames is reached – full lane recalculation is provided before the first success.

Such checks are provided:

- Radius of curvature shouldn’t be too small
- Distance between two lanes should be logical (more than 3 meters at least)
- Lanes are not intersected

All this checks are provided in method ***Line::check_correctness***

Discussion

The main issue I still faced is correct lane recognition in binary image. Especially I can see it on challenged videos. I think sanity check is also needed to be improved. Curve difference should be considered through this check.

One more idea to improve the quality is to add Hough lines to binaries additionally. In such case stricter filtering may be used, but lines will be added to the results. Filters threshold and combination also not ideal and are worth to be improved.

The other idea is to add addition mask polygon to the center of the image to avoid false positives lanes recognition which is especially critical for lanes histogram calculation

Some bugs also may be presented – especially on choosing lanes on frame step (at least some issue with intersection identified on harder_challenge_video.mp4).