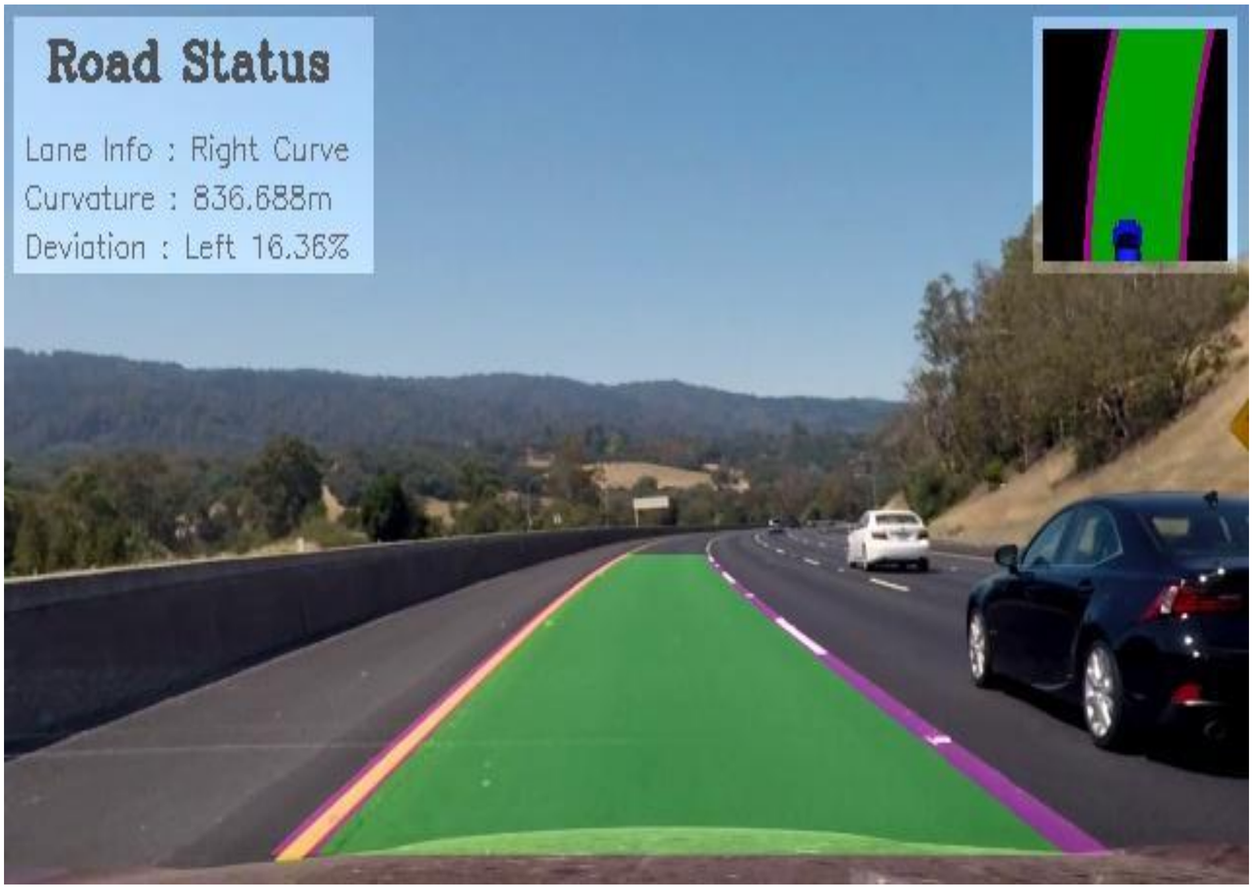
출처: https://github.com/windowsub0406/SelfDrivingCarND/blob/master/SDC_project_4

Advanced Lane Finding Project



result image(watch the full video below)

Introduction

This is **Advanced lane finding project** of Udacity's Self-Driving Car Engineering Nanodegree. We already completed lane finding project in the first project. In that project, we could find lane lines and made robust algorighm for shadow and some of occlusion. It might be enough in the straight highway.
But there are many curve lines in the road and that's why we need to detect curve lanes. In this project we'll find lane lines more specifically with computer vision.

이것은Udacity's Self-Driving Car Engineering Nanodegree의 Advanced lane finding

project입니다. 우리는 이미 첫 번째 프로젝트에서 차선 찾기 프로젝트를 완료했습니다. 그 프로젝트에서, 우리는 차선을

> 발견 할 수 있었고 그림자와 약간의 오클루젼을 위한 강력한 알고리즘을 만들 수 있었습니다. 그것은 straight(직선) 고속도로 에서 충분할지도 모른다. 그러나 도로에는 많은 곡선이 있으므로 곡선 차선을 감지해야 합니다. 이 프로젝트에서 우리는 컴퓨터 비전과 관련된 차선을 더욱 자세하게 찾을 것입니다.

Environment

software

> Windows 10(x64), Python 3.5, OpenCV 3.1.0

Files

main.py : main code
calibration.py : get calibration matrix
threshold.py : sobel edge & hls color
finding_lines.py : find & draw lane lines with sliding widow search
finding_lines_w.py : find & draw lane lines with sliding window search using weighted average method (for the challenge_video)

weighted average method를 사용하여, 슬라이딩 윈도우 검색으로 차선을 찾고 그립니다.

##The goals / steps of this project are the following:

이 프로젝트의 목표 및 단계

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.

  일련의 chessboard 이미지가 주어지면 camera calibration matrix 와 distortion coefficients(왜곡 계수)를 계산하라.

- Apply a distortion correction to raw images.

  Raw 이미지에 distortion correction(왜곡 보정) 을 적용한다.

- Use color transforms, gradients, etc., to create a thresholded binary image.

  color transforms, gradients 등을 사용하여, thresholded binary image(임계값 이진 이미지)를 만든다.

- Apply a perspective transform to rectify binary image ("birds-eye view").

  바이너리(이진) 영상을 수정하기 위해 perspective transform(원근감 변환)을 적용한다.

- Detect lane pixels and fit to find the lane boundary.

  차선 픽셀을 감지하고, 차선 경계를 찾으십시오.

- Determine the curvature of the lane and vehicle position with respect to center.

  중앙을 기준으로 차선 및 차량 위치의 곡률을 결정한다.

- Warp the detected lane boundaries back onto the original image.

  감지된 차선 경계를 원래 이미지로 되돌린다.

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

  차선 경계와 차선 곡률 및 차량 위치의 수치 추정을 시각적으로 표시한다.

##Camera Calibration

####When a camera looks at 3D objects in the real world and transforms them into a 2D image, it's not perfect because of a distortion. And the distortion brings an erroneous information.(e.g. changed object shape, bent lane lines)
So, we have to undo the distortion for getting useful data.

카메라가 실제 세계의 3D 물체를 보고 2D 이미지로 변환하면 왜곡 때문에 완벽하지 않습니다. 그리고 왜곡은 잘못된 정보를 가져옵니다 (예 : 개체 모양 변경, 구부러진 차선).

그래서 유용한 데이터를 얻기 위해 왜곡을 undo(되돌려야) 합니다.

The code for camera calibration step is contained in the calibration.py.

카메라 보정 단계 코드는 **calibration.py** 에 포함되어 있습니다.

I compute the camera matrix(intrinsic parameters) and distortion coefficients using the cv2.calibrateCamera() function with 20 9*6 sized chessboard images.

나는 20 9 * 6 크기의 체스 보드 이미지와 함께 **cv2.calibrateCamera()** 함수를 사용하여 카메라 행렬 (내장 매개 변수) 및 왜곡 계수를 계산합니다.

And applied this distortion correction to the test image using the cv2.undistort() function.

그리고 이 왜곡 보정을 **cv2.undistort()** 함수를 사용하여 테스트 이미지에 적용했습니다.

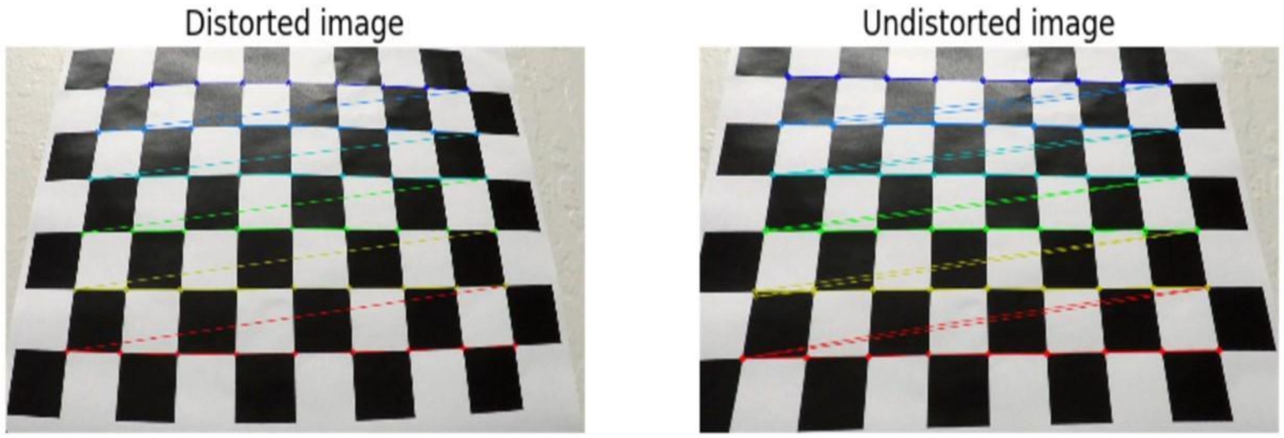
소스코드 (파이썬) : **calibration.py**

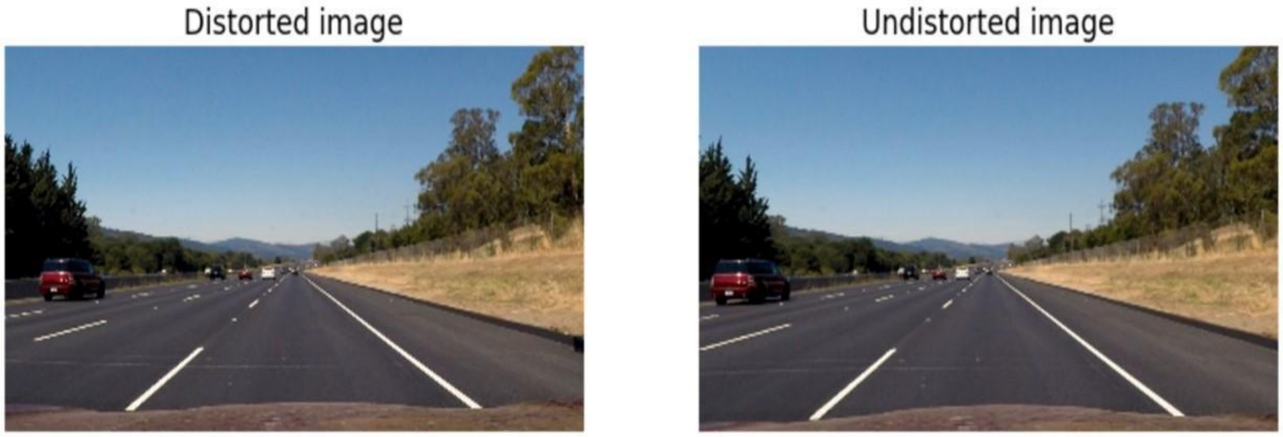
```
import numpy as np
import cv2
import matplotlib.image as mpimg
import glob

# Read in and make a list of calibration images
images = glob.glob('camera_cal/calibration*.jpg')

# Array to store object points and image points from all the images
```

```
objpoints = []  # 3D points in real world space
imgpoints = []  # 2D points in image plane

def calib():
    """
    To get an undistorted image, we need camera matrix & distortion coefficient
    Calculate them with 9*6 20 chessboard images
    """
    # Prepare object points
    objp = np.zeros((6 * 9, 3), np.float32)
    objp[:, :2] = np.mgrid[0:9, 0:6].T.reshape(-1, 2)  # x,y coordinates

    for fname in images:

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

        # Find the chessboard corners
        ret, corners = cv2.findChessboardCorners(gray, (9, 6), None)

        # If corners are found, add object points, image points
        if ret == True:
            imgpoints.append(corners)
            objpoints.append(objp)
        else:
            continue

    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],
None, None)

    return mtx, dist

def undistort(img, mtx, dist):
    """ undistort image """
    return cv2.undistort(img, mtx, dist, None, mtx)
```

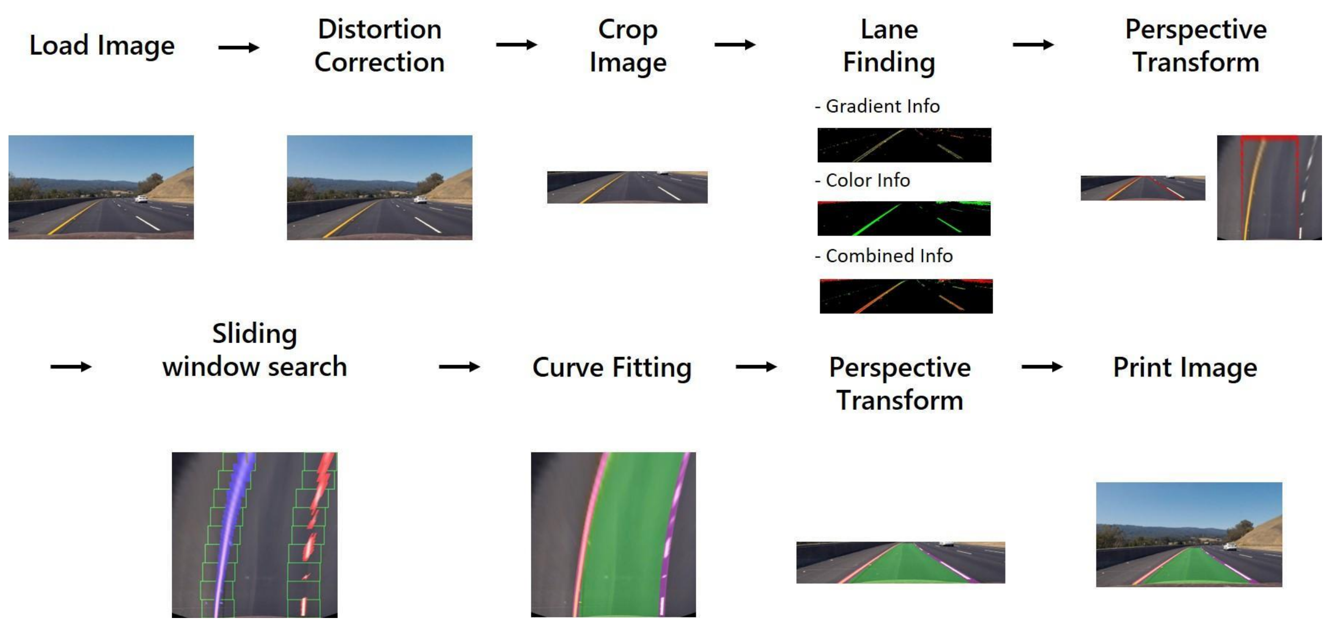
If an image loaded, we immediately undo distortion of the image using calculated calibration information.

이미지가 로드되면 계산된 캘리브레이션 정보를 사용하여 이미지의 왜곡을 즉시 undo합니다.


Distorted image
Undistorted image

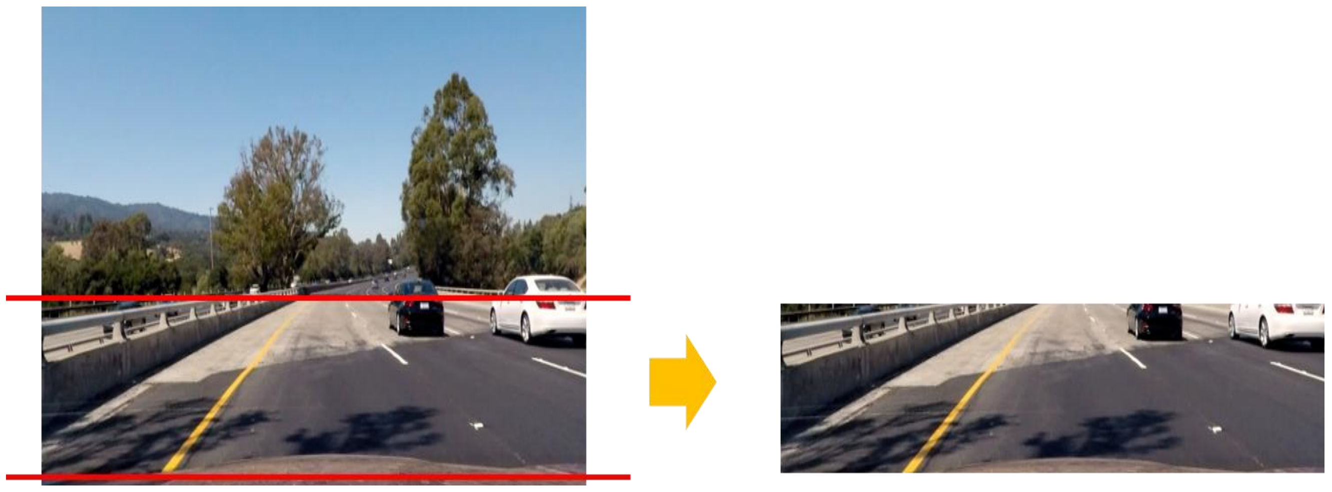

Distorted image
Undistorted image

##Pipeline



General Process

###1. Crop Image



In image, a bonnet and background are not necessary to find lane lines. Therefore, I cropped the inconsequential parts.

이미지에서 보닛과 배경은 차선을 찾는 데 필요하지 않습니다. 그러므로, 나는 중요하지 않은 부분을 자른다.

###2. Lane Finding

I used two approaches to find lane lines.

차선을 찾기 위해 두 가지 방법을 사용했습니다.

a **Gradient** approach and a **Color** approach. The code for lane finding step is contained in the **threshold.py**.

그라디언트 방식 및 컬러 방식을 지원합니다. 차선 찾기 단계를 위한 코드는 **threshold.py**에 포함되어 있습니다.

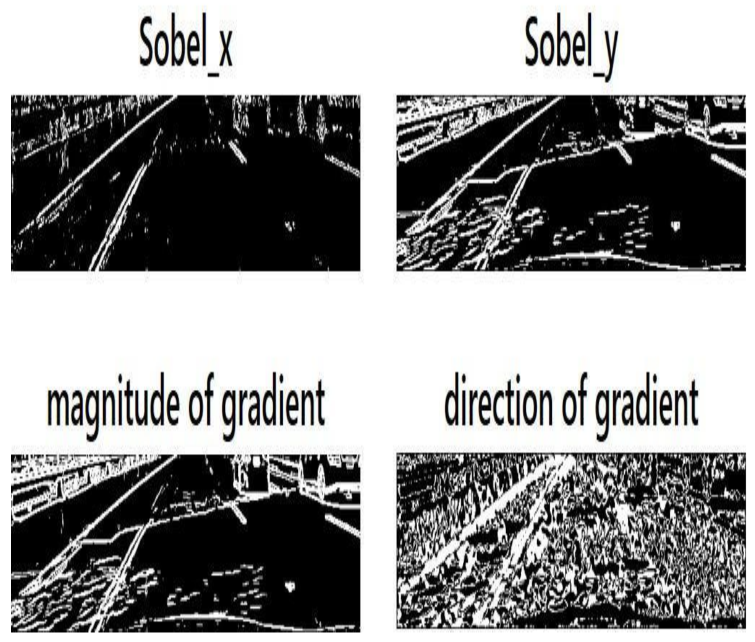
In gradient approach, I applied Sobel operator in the x, y directions. And calculated magnitude of the gradient in both the x and y directions and direction of the gradient. I used red channel of RGB instead of grayscaled image.

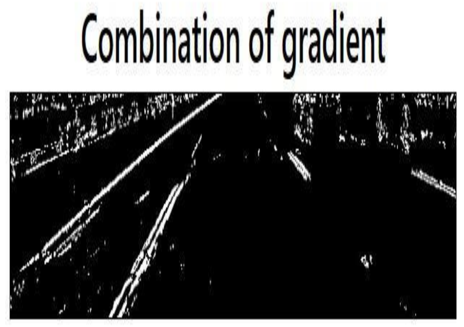
그래디언트 접근법에서, 저는 x, y 방향으로 소벨(Sobel) 연산자를 적용했습니다. 그리고 x와 y 방향 모두에서

그래디언트의 계산된 크기와 그래디언트의 방향. 나는 그레이 스케일 된 이미지 대신 RGB의 RED 채널을 사용했다.

And I combined them based on this code : 그리고 이 코드를 기반으로 결합했다:

```
gradient_comb[((sobelx>1) & (mag_img>1) & (dir_img>1)) | ((sobelx>1) & (sobely>1))] = 255
```





In Color approach, I used red channel of RGB Color space and H,L,S channel of HSV Color space. Red color(255,0,0) is included in white(255,255,255) and yellow(255,255,0) color. That's way I used it. Also I used HLS Color space because we could be robust in brightness.

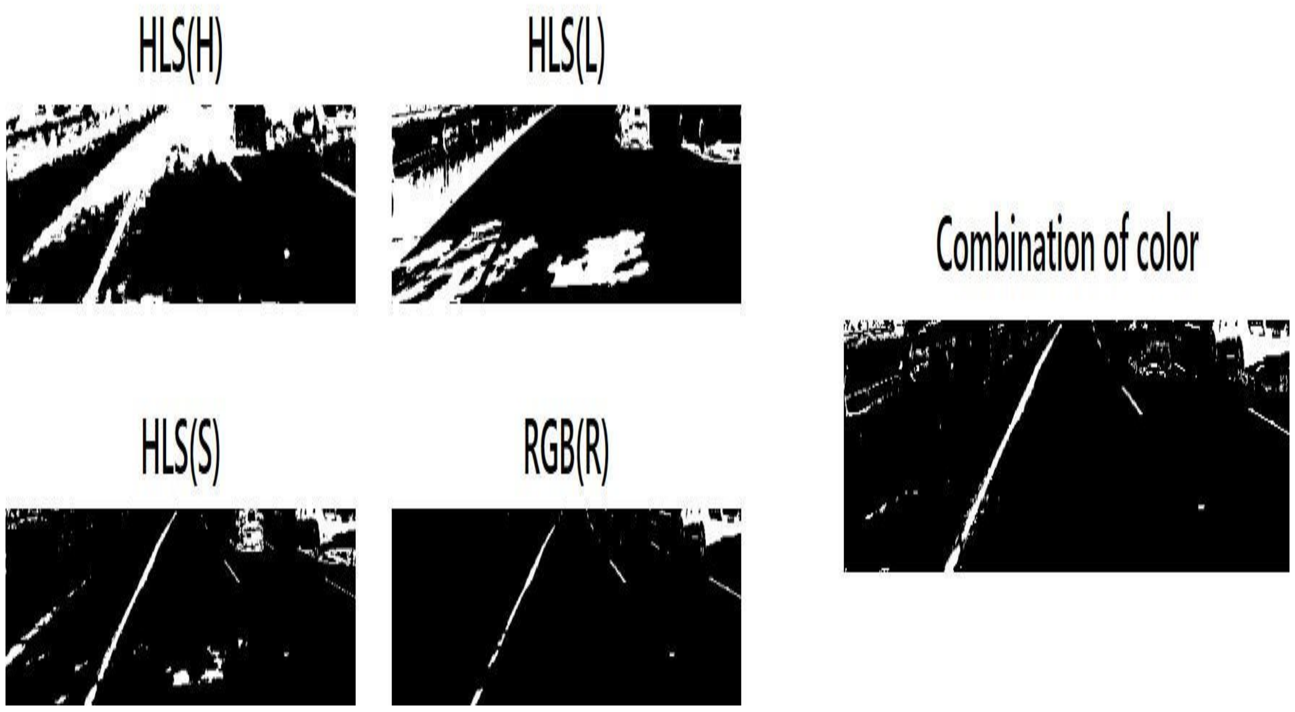
Color 방식에서는 RGB 색상 공간의 RED 채널과 HSV 색상 공간의 H, L, S 채널을 사용했습니다. 빨간색 (255,0,0)은 흰색(255,255,255) 및 노란색(255,255,0) 색상에 포함됩니다. 그렇게 사용했습니다. 또한 우리는 밝기가 강하기 때문에 HLS Color Space를 사용했습니다.

I combined them based on this code : 그리고 이 코드를 기반으로 이것들을 결합했다

```
: hls_comb[((s_img>1) & (l_img == 0)) | ((s_img==0) & (h_img>1) & (l_img>1)) | (R>1)] = 255
```

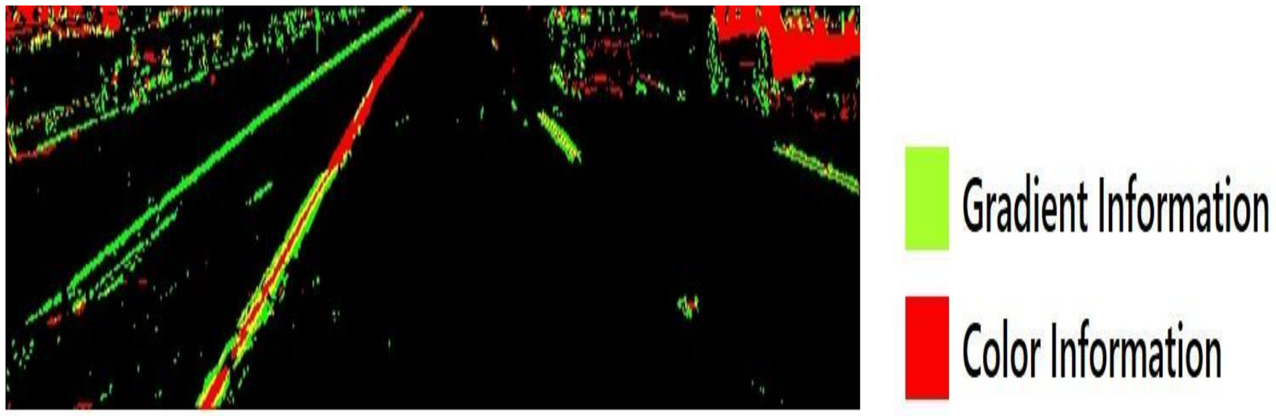
With this method, I could eliminate unnecessary shadow information.

이 방법을 사용하면 불필요한 음영 정보를 제거 할 수 있습니다.



This is combination of color and gradient thresholds.

이것 색상 및 그래디언트 임계값의 조합입니다.



소스코드 (파이썬) : threshold.py

```
import numpy as np
import cv2

def sobel_xy(img, orient='x', thresh=(20, 100)):
    """
    Define a function that applies Sobel x or y.
    The gradient in the x-direction emphasizes edges closer to vertical.
    The gradient in the y-direction emphasizes edges closer to horizontal.
    """
    # img = exposure.equalize_hist(img)
    # adaptive histogram equalization
    # img = exposure.equalize_adapthist(img, clip_limit=0.01)

    if orient == 'x':
        abs_sobel = np.absolute(cv2.Sobel(img, cv2.CV_64F, 1, 0))
    if orient == 'y':
        abs_sobel = np.absolute(cv2.Sobel(img, cv2.CV_64F, 0, 1))
    # Rescale back to 8 bit integer
    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])] = 255

    # Return the result
    return binary_output

def mag_thresh(img, sobel_kernel=3, mag_thresh=(0, 255)):
    """
    Define a function to return the magnitude of the gradient
    for a given sobel kernel size and threshold values
    """

    # adaptive histogram equalization
    # img = exposure.equalize_adapthist(img, clip_limit=0.01)
    # Take both Sobel x and y gradients
    sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    # Calculate the gradient magnitude
    gradmag = np.sqrt(sobelx**2 + sobely**2)
    # Rescale to 8 bit
    scale_factor = np.max(gradmag)/255
    gradmag = (gradmag/scale_factor).astype(np.uint8)
    # Create a binary image of ones where threshold is met, zeros otherwise
    binary_output = np.zeros_like(gradmag)
    binary_output[(gradmag >= mag_thresh[0]) & (gradmag <= mag_thresh[1])] = 255

    # Return the binary image
    return binary_output

def dir_thresh(img, sobel_kernel=3, thresh=(0.7, 1.3)):
    """
    computes the direction of the gradient
    """
    # Calculate the x and y gradients
    sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    # Take the absolute value of the gradient direction,
    # apply a threshold, and create a binary image result
    absgraddir = np.arctan2(np.absolute(sobely), np.absolute(sobelx))
    binary_output = np.zeros_like(absgraddir)
    binary_output[(absgraddir >= thresh[0]) & (absgraddir <= thresh[1])] = 255
    # Return the binary image
    return binary_output.astype(np.uint8)

def ch_thresh(ch, thresh=(80, 255)):
    binary = np.zeros_like(ch)
    binary[(ch > thresh[0]) & (ch <= thresh[1])] = 255
    return binary

def gradient_combine(img, th_x, th_y, th_mag, th_dir):
    """
    Find lane lines with gradient information of Red channel
    """
    rows, cols = img.shape[:2]
    R = img[230:rows - 12, 0:cols, 2]

    sobelx = sobel_xy(R, 'x', th_x)
    #cv2.imshow('sobel_x', sobelx)
    sobely = sobel_xy(R, 'y', th_y)
    #cv2.imshow('sobel_y', sobely)
    mag_img = mag_thresh(R, 3, th_mag)
    #cv2.imshow('sobel_mag', mag_img)
    dir_img = dir_thresh(R, 15, th_dir)
    #cv2.imshow('result5', dir_img)

    # combine gradient measurements
    gradient_comb = np.zeros_like(dir_img).astype(np.uint8)
    gradient_comb[((sobelx > 1) & (mag_img > 1) & (dir_img > 1)) | ((sobelx > 1) & (sobely > 1))] = 
255

    return gradient_comb

def hls_combine(img, th_h, th_l, th_s):
    # convert to hls color space
    hls = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)

    rows, cols = img.shape[:2]
    R = img[230:rows - 12, 0:cols, 2]
    _, R = cv2.threshold(R, 180, 255, cv2.THRESH_BINARY)
    #cv2.imshow('red!!!',R)
    H = hls[230:rows - 12, 0:cols, 0]
    L = hls[230:rows - 12, 0:cols, 1]
    S = hls[230:rows - 12, 0:cols, 2]

    h_img = ch_thresh(H, th_h)
    #cv2.imshow('HLS (H) threshold', h_img)
    l_img = ch_thresh(L, th_l)
    #cv2.imshow('HLS (L) threshold', l_img)
    s_img = ch_thresh(S, th_s)
    #cv2.imshow('HLS (S) threshold', s_img)

    # Two cases - lane lines in shadow or not
    hls_comb = np.zeros_like(s_img).astype(np.uint8)
    hls_comb[((s_img > 1) & (l_img == 0)) | ((s_img == 0) & (h_img > 1) & (l_img > 1)) | (R > 1)] = 255
    #hls_comb[((s_img > 1) & (h_img > 1)) | (R > 1)] = 255
    return hls_comb

def comb_result(grad, hls):
    """ give different value to distinguish them """
    result = np.zeros_like(hls).astype(np.uint8)
    #result[((grad > 1) | (hls > 1))] = 255
    result[(grad > 1)] = 100
    result[(hls > 1)] = 255

    return result
```

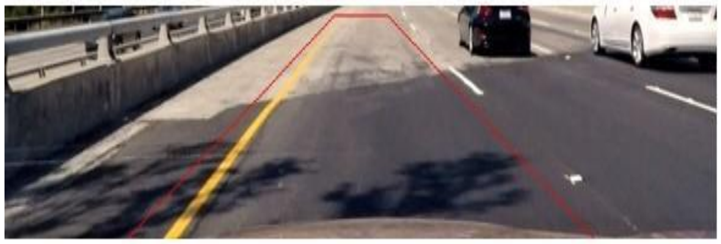
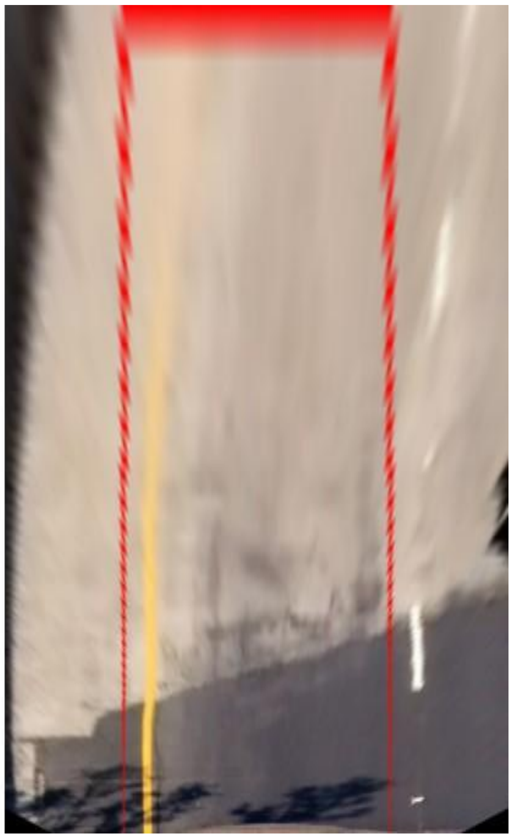
###3. Perspective Transform

원근감 변환

We can assume the road is a flat plane. Pick 4 points of straight lane lines and apply perspective transform to the lines look straight. It is also called Bird's eye view.

우리는 도로가 flat plane(평평한 비행기??)라고 생각할 수 있습니다. 직선 차선 4 점을 선택하고 직선으로 원근감

변환을 적용하십시오. Bird 's eye view라고도합니다.

###4. Sliding Window Search

슬라이딩 창 검색

The code for Sliding window search is contained in the `finding_lines.py` or `finding_lines_w.py`.

슬라이딩 창 검색 코드는**finding_lines.py** 또는**finding_lines_w.py**에 있습니다.

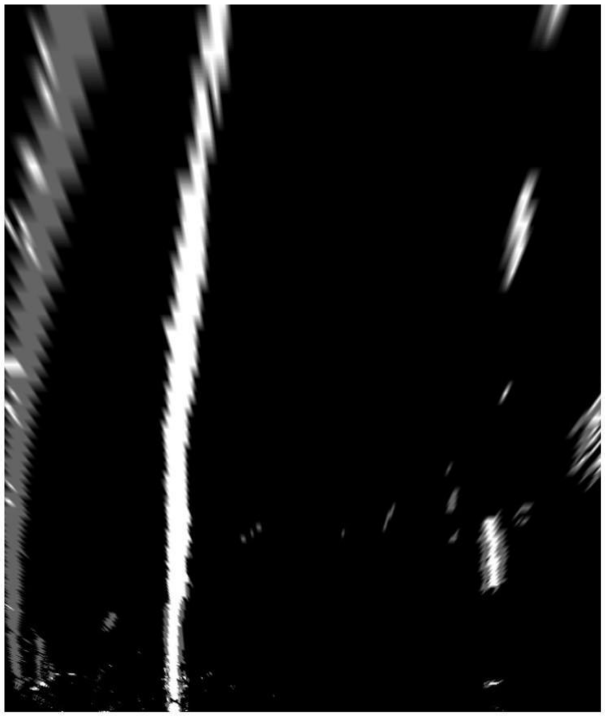
In the video, we could predict the position of lane lines by checking previous frame's information. But we need an other method for a first frame.

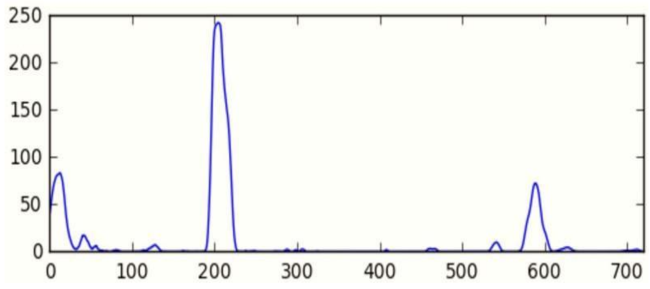
비디오에서 이전 프레임의 정보를 확인하여 차선 위치를 예측할 수 있었습니다. 그러나 첫 번째 프레임에 대해 다른 방법이 필요합니다.

In my code, if the frame is first frame or lost lane position, found first window position using histogram. Just accumulated non-zero pixels along the columns in the lower 2/3 of the image.

내 코드에서 프레임이 첫 프레임이거나 손실된 차선 위치이면 히스토그램을 사용하여 첫 번째 창 위치를 찾았습니다.

이미지의 하단 2/3에있는 열을 따라 0이 아닌 픽셀을 누적했습니다.

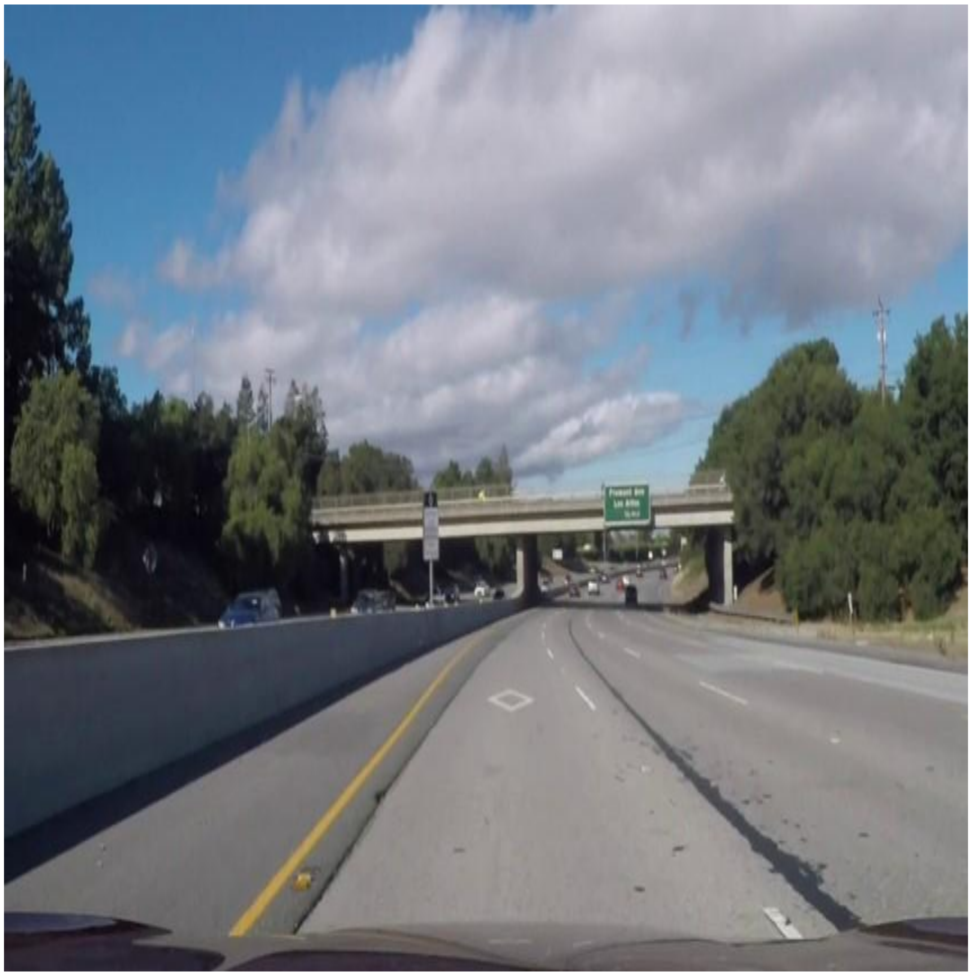




In the course, we estimated curve line by using all non-zero pixels of windows. Non-zero piexels include **color information** and **gradient information** in bird's eyes view binary image. It works well in `project_video`.

이 과정에서 windows(창)의 0이 아닌 픽셀을 모두 사용하여 곡선을 추정했습니다. 0이 아닌 piexels은 조감도보기 바이너리 이미지에서 색 정보와 그라디언트 정보를 포함합니다. 그것은 project_video에서 잘 작동합니다.

But it has a problem. 하지만 문제가 있습니다.



This is one frame of challenge_video.

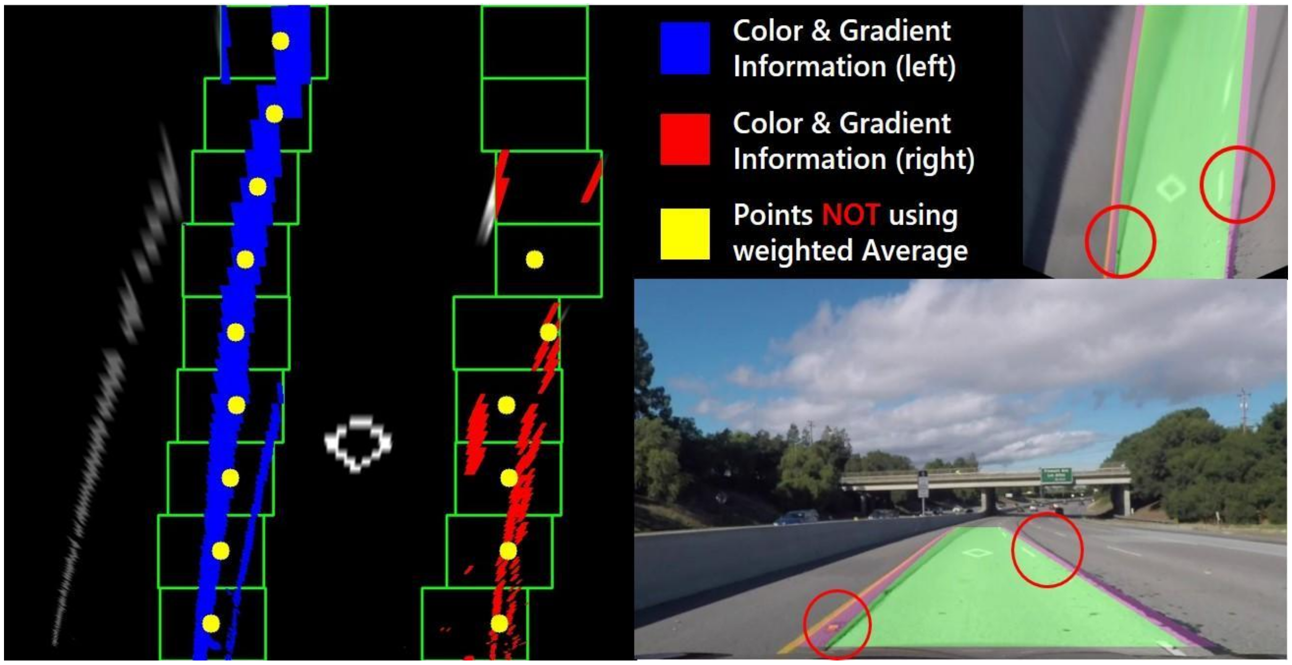
In this image, there are some cracks and dark trails near the lane lines. Let's check the result.

If we fit curve lines with non-zero pixels, the result is here.

이것은 challenge_video의 한 프레임입니다.

이 이미지에는 차선 근처에 균열과 어두운 흔적이 있습니다. 결과를 확인해 봅시다.

0이 아닌 픽셀로 곡선을 맞추면 결과가 나타납니다.


Color & Gradient Information (left)
Color & Gradient Information (right)
Points NOT using weighted Average

As you can see, we couldn't detect exact lane positions. Because our gradient information have cracks information and it occurs error of position.

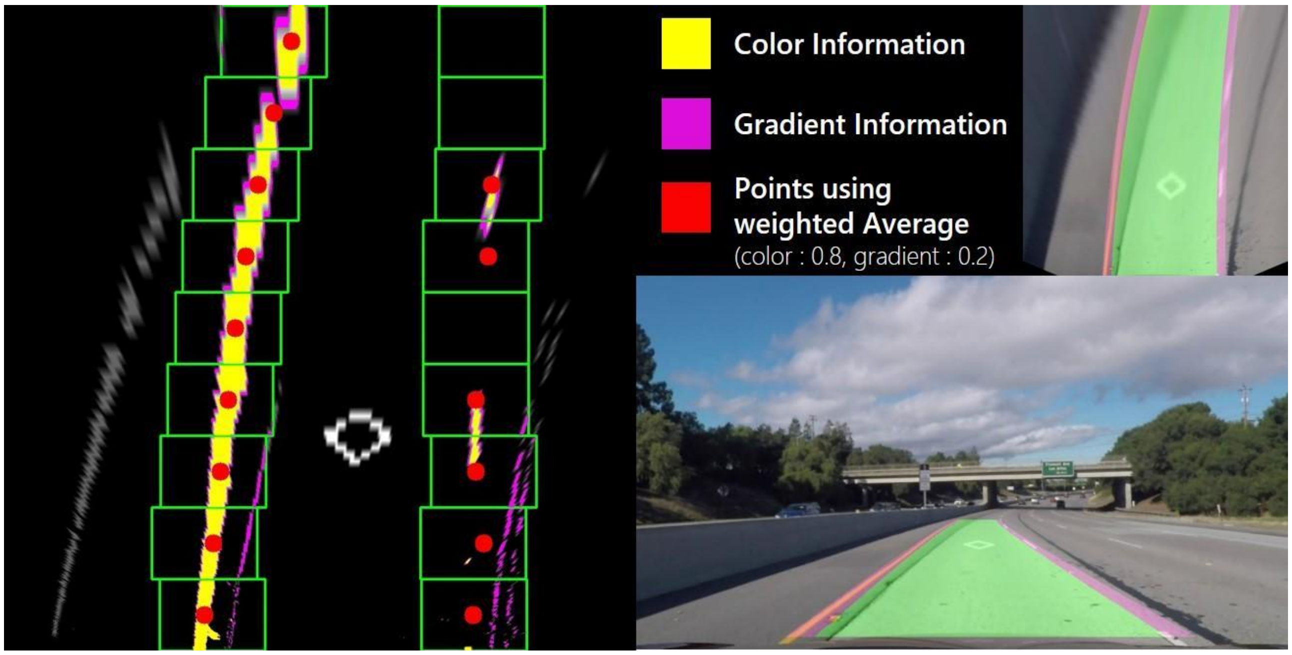
보시다시피 정확한 차선 위치를 감지할 수 없었습니다. 그라디언트 정보에 균열 정보가 있고 위치 오류가 발생하기

때문입니다.

So, I used **weighted average** method. I put **0.8** weight value to color information and **0.2** to gradient information. And calculated x-average by using weighted average in the window. This is the result.

그래서 가중 평균법을 사용했습니다. 색상 정보에 0.8의 가중치를, 그래디언트 정보에 0.2를 넣었습니다. 창에서 가중

평균을 사용하여 x 평균을 계산했습니다. 이것이 결과입니다.


🟨 Color Information
🟪 Gradient Information
🟥 Points using weighted Average (color : 0.8, gradient : 0.2)

<u>소스코드(파이썬)</u> : **finding_lines.py**

```
import numpy as np
import cv2
from PIL import Image
import matplotlib.image as mpimg

class Line:
    def __init__(self):
        # was the line detected in the last iteration?
        self.detected = False
        # Set the width of the windows +/- margin
        self.window_margin = 56
        # x values of the fitted line over the last n iterations
        self.prevx = []
        # polynomial coefficients for the most recent fit
        self.current_fit = [np.array([False])]
        #radius of curvature of the line in some units
        self.radius_of_curvature = None
        # starting x_value
        self.startx = None
        # ending x_value
        self.endx = None
        # x values for detected line pixels
        self.allx = None
        # y values for detected line pixels
        self.ally = None
        # road information
        self.road_inf = None
        self.curvature = None
        self.deviation = None

def warp_image(img, src, dst, size):
    """ Perspective Transform """
    M = cv2.getPerspectiveTransform(src, dst)
    Minv = cv2.getPerspectiveTransform(dst, src)
    warp_img = cv2.warpPerspective(img, M, size, flags=cv2.INTER_LINEAR)

    return warp_img, M, Minv

def rad_of_curvature(left_line, right_line):
    """ measure radius of curvature  """

    ploty = left_line.ally
    leftx, rightx = left_line.allx, right_line.allx

    leftx = leftx[::-1]  # Reverse to match top-to-bottom in y
    rightx = rightx[::-1]  # Reverse to match top-to-bottom in y

    # Define conversions in x and y from pixels space to meters
    width_lanes = abs(right_line.startx - left_line.startx)
    ym_per_pix = 30 / 720  # meters per pixel in y dimension
    xm_per_pix = 3.7*(720/1280) / width_lanes  # meters per pixel in x dimension

    # Define y-value where we want radius of curvature
    # the maximum y-value, corresponding to the bottom of the image
    y_eval = np.max(ploty)

    # Fit new polynomials to x,y in world space
    left_fit_cr = np.polyfit(ploty * ym_per_pix, leftx * xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty * 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]) ** 2) ** 1.5) /
np.absolute(
        2 * left_fit_cr[0])
    right_curverad = ((1 + (2 * right_fit_cr[0] * y_eval * ym_per_pix + right_fit_cr[1]) ** 2) ** 1.5) /
np.absolute(
        2 * right_fit_cr[0])
    # radius of curvature result
    left_line.radius_of_curvature = left_curverad
    right_line.radius_of_curvature = right_curverad

def smoothing(lines, pre_lines=3):
    # collect lines & print average line
    lines = np.squeeze(lines)
    avg_line = np.zeros((720))

    for ii, line in enumerate(reversed(lines)):
        if ii == pre_lines:
            break
        avg_line += line
    avg_line = avg_line / pre_lines

    return avg_line
```

```
def blind_search(b_img, left_line, right_line):
    """
    blind search - first frame, lost lane lines
    using histogram & sliding window
    """
    # Take a histogram of the bottom half of the image
    histogram = np.sum(b_img[int(b_img.shape[0] / 2):, :], axis=0)

    # Create an output image to draw on and  visualize the result
    output = np.dstack((b_img, b_img, b_img)) * 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)
    start_leftX = np.argmax(histogram[:midpoint])
    start_rightX = np.argmax(histogram[midpoint:]) + midpoint

    # Choose the number of sliding windows
    num_windows = 9
    # Set height of windows
    window_height = np.int(b_img.shape[0] / num_windows)

    # Identify the x and y positions of all nonzero pixels in the image
    nonzero = b_img.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])

    # Current positions to be updated for each window
    current_leftX = start_leftX
    current_rightX = start_rightX

    # Set minimum number of pixels found to recenter window
    min_num_pixel = 50
    # Create empty lists to receive left and right lane pixel indices
    win_left_lane = []
    win_right_lane = []

    window_margin = left_line.window_margin

    # Step through the windows one by one
    for window in range(num_windows):
        # Identify window boundaries in x and y (and right and left)
        win_y_low = b_img.shape[0] - (window + 1) * window_height
        win_y_high = b_img.shape[0] - window * window_height
        win_leftx_min = current_leftX - window_margin
        win_leftx_max = current_leftX + window_margin
        win_rightx_min = current_rightX - window_margin
        win_rightx_max = current_rightX + window_margin

        # Draw the windows on the visualization image
        cv2.rectangle(output, (win_leftx_min, win_y_low), (win_leftx_max, win_y_high), (0, 255,
0), 2)
        cv2.rectangle(output, (win_rightx_min, win_y_low), (win_rightx_max, win_y_high), (0,
255, 0), 2)

        # Identify the nonzero pixels in x and y within the window
        left_window_inds = ((nonzeroy >= win_y_low) & (nonzeroy <= win_y_high) & (nonzerox >=
win_leftx_min) & (
            nonzerox <= win_leftx_max)).nonzero()[0]
        right_window_inds = ((nonzeroy >= win_y_low) & (nonzeroy <= win_y_high) & (nonzerox
>= win_rightx_min) & (
            nonzerox <= win_rightx_max)).nonzero()[0]
        # Append these indices to the lists
        win_left_lane.append(left_window_inds)
        win_right_lane.append(right_window_inds)
        # If you found > minpix pixels, recenter next window on their mean position
        if len(left_window_inds) > min_num_pixel:
            current_leftX = np.int(np.mean(nonzerox[left_window_inds]))
        if len(right_window_inds) > min_num_pixel:
            current_rightX = np.int(np.mean(nonzerox[right_window_inds]))

    # Concatenate the arrays of indices
    win_left_lane = np.concatenate(win_left_lane)
    win_right_lane = np.concatenate(win_right_lane)

    # Extract left and right line pixel positions
    leftx, lefty = nonzerox[win_left_lane], nonzeroy[win_left_lane]
    rightx, righty = nonzerox[win_right_lane], nonzeroy[win_right_lane]

    output[lefty, leftx] = [255, 0, 0]
    output[righty, rightx] = [0, 0, 255]

    # Fit a second order polynomial to each
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)

    left_line.current_fit = left_fit
    right_line.current_fit = right_fit

    # Generate x and y values for plotting
    ploty = np.linspace(0, b_img.shape[0] - 1, b_img.shape[0])

    # ax^2 + bx + c
    left_plotx = left_fit[0] * ploty ** 2 + left_fit[1] * ploty + left_fit[2]
    right_plotx = right_fit[0] * ploty ** 2 + right_fit[1] * ploty + right_fit[2]

    left_line.prevx.append(left_plotx)
```

```
    right_line.prevx.append(right_plotx)

    if len(left_line.prevx) > 10:
        left_avg_line = smoothing(left_line.prevx, 10)
        left_avg_fit = np.polyfit(ploty, left_avg_line, 2)
        left_fit_plotx = left_avg_fit[0] * ploty ** 2 + left_avg_fit[1] * ploty + left_avg_fit[2]
        left_line.current_fit = left_avg_fit
        left_line.allx, left_line.ally = left_fit_plotx, ploty
    else:
        left_line.current_fit = left_fit
        left_line.allx, left_line.ally = left_plotx, ploty

    if len(right_line.prevx) > 10:
        right_avg_line = smoothing(right_line.prevx, 10)
        right_avg_fit = np.polyfit(ploty, right_avg_line, 2)
        right_fit_plotx = right_avg_fit[0] * ploty ** 2 + right_avg_fit[1] * ploty + right_avg_fit[2]
        right_line.current_fit = right_avg_fit
        right_line.allx, right_line.ally = right_fit_plotx, ploty
    else:
        right_line.current_fit = right_fit
        right_line.allx, right_line.ally = right_plotx, ploty

    left_line.startx, right_line.startx = left_line.allx[len(left_line.allx)-1],
right_line.allx[len(right_line.allx)-1]
    left_line.endx, right_line.endx = left_line.allx[0], right_line.allx[0]

    left_line.detected, right_line.detected = True, True
    # print radius of curvature
    rad_of_curvature(left_line, right_line)
    return output

def prev_window_refer(b_img, left_line, right_line):
    """
    refer to previous window info - after detecting lane lines in previous frame
    """
    # Create an output image to draw on and  visualize the result
    output = np.dstack((b_img, b_img, b_img)) * 255

    # Identify the x and y positions of all nonzero pixels in the image
    nonzero = b_img.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])

    # Set margin of windows
    window_margin = left_line.window_margin

    left_line_fit = left_line.current_fit
    right_line_fit = right_line.current_fit
    leftx_min = left_line_fit[0] * nonzeroy ** 2 + left_line_fit[1] * nonzeroy + left_line_fit[2] -
window_margin
    leftx_max = left_line_fit[0] * nonzeroy ** 2 + left_line_fit[1] * nonzeroy + left_line_fit[2] +
window_margin
    rightx_min = right_line_fit[0] * nonzeroy ** 2 + right_line_fit[1] * nonzeroy + right_line_fit[2] -
window_margin
    rightx_max = right_line_fit[0] * nonzeroy ** 2 + right_line_fit[1] * nonzeroy + right_line_fit[2] +
window_margin

    # Identify the nonzero pixels in x and y within the window
    left_inds = ((nonzerox >= leftx_min) & (nonzerox <= leftx_max)).nonzero()[0]
    right_inds = ((nonzerox >= rightx_min) & (nonzerox <= rightx_max)).nonzero()[0]

    # Extract left and right line pixel positions
    leftx, lefty = nonzerox[left_inds], nonzeroy[left_inds]
    rightx, righty = nonzerox[right_inds], nonzeroy[right_inds]

    output[lefty, leftx] = [255, 0, 0]
```

```
output[righty, rightx] = [0, 0, 255]

# 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, b_img.shape[0] - 1, b_img.shape[0])

# ax^2 + bx + c
left_plotx = left_fit[0] * ploty ** 2 + left_fit[1] * ploty + left_fit[2]
right_plotx = right_fit[0] * ploty ** 2 + right_fit[1] * ploty + right_fit[2]

leftx_avg = np.average(left_plotx)
rightx_avg = np.average(right_plotx)

left_line.prevx.append(left_plotx)
right_line.prevx.append(right_plotx)

if len(left_line.prevx) > 10:
    left_avg_line = smoothing(left_line.prevx, 10)
    left_avg_fit = np.polyfit(ploty, left_avg_line, 2)
    left_fit_plotx = left_avg_fit[0] * ploty ** 2 + left_avg_fit[1] * ploty + left_avg_fit[2]
    left_line.current_fit = left_avg_fit
    left_line.allx, left_line.ally = left_fit_plotx, ploty
else:
    left_line.current_fit = left_fit
    left_line.allx, left_line.ally = left_plotx, ploty

if len(right_line.prevx) > 10:
    right_avg_line = smoothing(right_line.prevx, 10)
    right_avg_fit = np.polyfit(ploty, right_avg_line, 2)
    right_fit_plotx = right_avg_fit[0] * ploty ** 2 + right_avg_fit[1] * ploty + right_avg_fit[2]
```

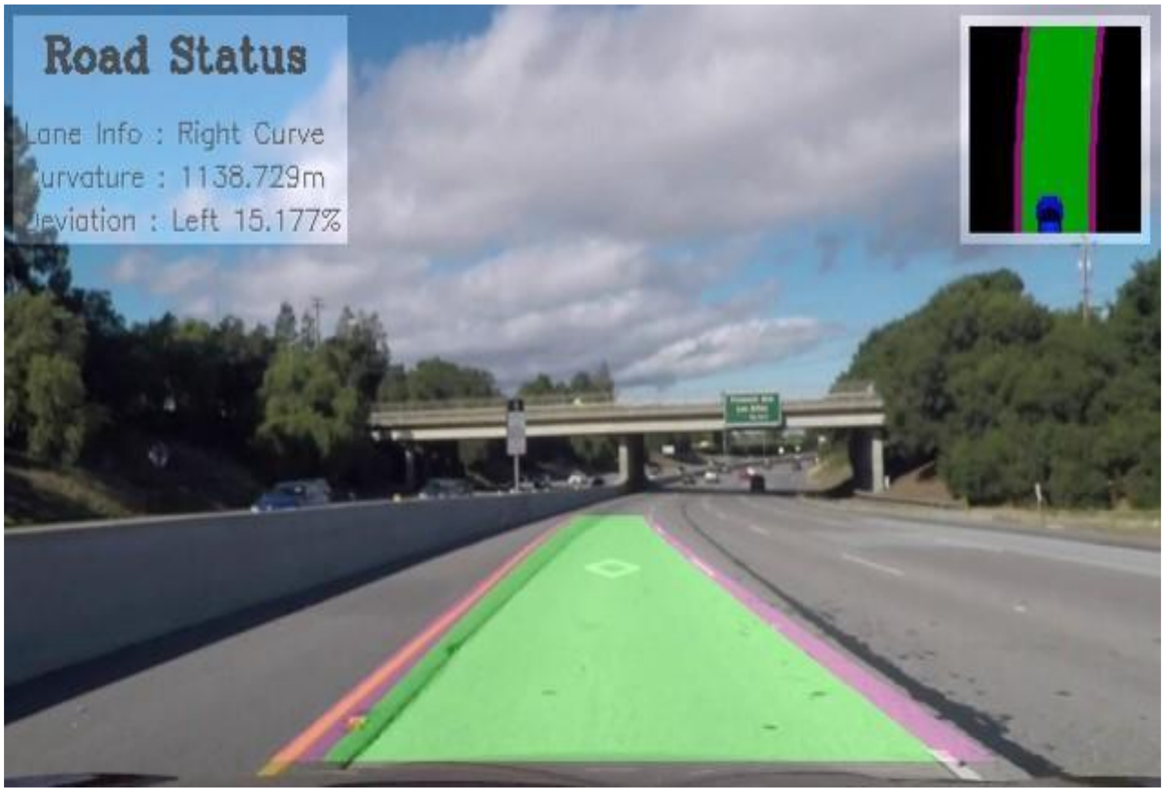
```
        right_line.current_fit = right_avg_fit
        right_line.allx, right_line.ally = right_fit_plotx, ploty
    else:
        right_line.current_fit = right_fit
        right_line.allx, right_line.ally = right_plotx, ploty

    # goto blind_search if the standard value of lane lines is high.
    standard = np.std(right_line.allx - left_line.allx)

    if (standard > 80):
        left_line.detected = False

    left_line.startx, right_line.startx = left_line.allx[len(left_line.allx) - 1],
right_line.allx[len(right_line.allx) - 1]
    left_line.endx, right_line.endx = left_line.allx[0], right_line.allx[0]

    # print radius of curvature
    rad_of_curvature(left_line, right_line)
    return output

def find_LR_lines(binary_img, left_line, right_line):
    """
    find left, right lines & isolate left, right lines
    blind search - first frame, lost lane lines
    previous window - after detecting lane lines in previous frame
    """

    # if don't have lane lines info
    if left_line.detected == False:
        return blind_search(binary_img, left_line, right_line)
    # if have lane lines info
    else:
        return prev_window_refer(binary_img, left_line, right_line)
```

```
def draw_lane(img, left_line, right_line, lane_color=(255, 0, 255), road_color=(0, 255, 0)):
    """ draw lane lines & current driving space """
    window_img = np.zeros_like(img)

    window_margin = left_line.window_margin
    left_plotx, right_plotx = left_line.allx, right_line.allx
    ploty = left_line.ally

    # Generate a polygon to illustrate the search window area
    # And recast the x and y points into usable format for cv2.fillPoly()
    left_pts_l = np.array([np.transpose(np.vstack([left_plotx - window_margin/5, ploty]))])
    left_pts_r = np.array([np.flipud(np.transpose(np.vstack([left_plotx + window_margin/5,
ploty])))])
    left_pts = np.hstack((left_pts_l, left_pts_r))
    right_pts_l = np.array([np.transpose(np.vstack([right_plotx - window_margin/5, ploty]))])
    right_pts_r = np.array([np.flipud(np.transpose(np.vstack([right_plotx + window_margin/5,
ploty])))])
    right_pts = np.hstack((right_pts_l, right_pts_r))

    # Draw the lane onto the warped blank image
    cv2.fillPoly(window_img, np.int_([left_pts]), lane_color)
    cv2.fillPoly(window_img, np.int_([right_pts]), lane_color)

    # Recast the x and y points into usable format for cv2.fillPoly()
    pts_left = np.array([np.transpose(np.vstack([left_plotx+window_margin/5, ploty]))])
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_plotx-window_margin/5,
ploty])))])
    pts = np.hstack((pts_left, pts_right))

    # Draw the lane onto the warped blank image
    cv2.fillPoly(window_img, np.int_([pts]), road_color)
    result = cv2.addWeighted(img, 1, window_img, 0.3, 0)
    return result, window_img

def road_info(left_line, right_line):
    """ print road information onto result image """
    curvature = (left_line.radius_of_curvature + right_line.radius_of_curvature) / 2

    direction = ((left_line.endx - left_line.startx) + (right_line.endx - right_line.startx)) / 2

    if curvature > 2000 and abs(direction) < 100:
        road_inf = 'No Curve'
        curvature = -1
    elif curvature <= 2000 and direction < - 50:
        road_inf = 'Left Curve'
    elif curvature <= 2000 and direction > 50:
        road_inf = 'Right Curve'
    else:
        if left_line.road_inf != None:
            road_inf = left_line.road_inf
            curvature = left_line.curvature
        else:
            road_inf = 'None'
            curvature = curvature

    center_lane = (right_line.startx + left_line.startx) / 2
    lane_width = right_line.startx - left_line.startx

    center_car = 720 / 2
    if center_lane > center_car:
        deviation = 'Left ' + str(round(abs(center_lane - center_car)/(lane_width / 2)*100, 3)) + '%'
    elif center_lane < center_car:
        deviation = 'Right ' + str(round(abs(center_lane - center_car)/(lane_width / 2)*100, 3)) +
'%'
```

```
    else:
        deviation = 'Center'
    left_line.road_inf = road_inf
    left_line.curvature = curvature
    left_line.deviation = deviation

    return road_inf, curvature, deviation

def print_road_status(img, left_line, right_line):
    """ print road status (curve direction, radius of curvature, deviation) """
    road_inf, curvature, deviation = road_info(left_line, right_line)
    cv2.putText(img, 'Road Status', (22, 30), cv2.FONT_HERSHEY_COMPLEX, 0.7, (80, 80, 80),
2)

    lane_inf = 'Lane Info : ' + road_inf
    if curvature == -1:
        lane_curve = 'Curvature : Straight line'
    else:
        lane_curve = 'Curvature : {0:0.3f}m'.format(curvature)
    deviate = 'Deviation : ' + deviation

    cv2.putText(img, lane_inf, (10, 63), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (100, 100, 100), 1)
    cv2.putText(img, lane_curve, (10, 83), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (100, 100, 100),
1)
    cv2.putText(img, deviate, (10, 103), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (100, 100, 100), 1)

    return img

def print_road_map(image, left_line, right_line):
    """ print simple road map """
    img = cv2.imread('images/top_view_car.png', -1)
    img = cv2.resize(img, (120, 246))
    rows, cols = image.shape[:2]
    window_img = np.zeros_like(image)

    window_margin = left_line.window_margin
    left_plotx, right_plotx = left_line.allx, right_line.allx
    ploty = left_line.ally
    lane_width = right_line.startx - left_line.startx
    lane_center = (right_line.startx + left_line.startx) / 2
    lane_offset = cols / 2 - (2*left_line.startx + lane_width) / 2
    car_offset = int(lane_center - 360)
    # Generate a polygon to illustrate the search window area
    # And recast the x and y points into usable format for cv2.fillPoly()
    left_pts_l = np.array([np.transpose(np.vstack([right_plotx + lane_offset - lane_width -
window_margin / 4, ploty]))])
    left_pts_r = np.array([np.flipud(np.transpose(np.vstack([right_plotx + lane_offset -
lane_width+ window_margin / 4, ploty])))])
    left_pts = np.hstack((left_pts_l, left_pts_r))
    right_pts_l = np.array([np.transpose(np.vstack([right_plotx + lane_offset - window_margin /
4, ploty]))])
    right_pts_r = np.array([np.flipud(np.transpose(np.vstack([right_plotx + lane_offset +
window_margin / 4, ploty])))])
    right_pts = np.hstack((right_pts_l, right_pts_r))

    # Draw the lane onto the warped blank image
    cv2.fillPoly(window_img, np.int_([left_pts]), (140, 0, 170))
    cv2.fillPoly(window_img, np.int_([right_pts]), (140, 0, 170))

    # Recast the x and y points into usable format for cv2.fillPoly()
    pts_left = np.array([np.transpose(np.vstack([right_plotx + lane_offset - lane_width +
window_margin / 4, ploty]))])
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_plotx + lane_offset -
window_margin / 4, ploty])))])
    pts = np.hstack((pts_left, pts_right))
```

```
# Draw the lane onto the warped blank image
cv2.fillPoly(window_img, np.int_([pts]), (0, 160, 0))

#window_img[10:133,300:360] = img
road_map = Image.new('RGBA', image.shape[:2], (0, 0, 0, 0))
window_img = Image.fromarray(window_img)
img = Image.fromarray(img)
road_map.paste(window_img, (0, 0))
road_map.paste(img, (300-car_offset, 590), mask=img)
road_map = np.array(road_map)
road_map = cv2.resize(road_map, (95, 95))
road_map = cv2.cvtColor(road_map, cv2.COLOR_BGRA2BGR)
return road_map
```

###6. Road information



In my output video, I included some road informations.

출력 비디오에는 도로 정보가 포함되어 있습니다.

####Lane Info 차선 정보

 estimate lane status that is a straight line, or left/right curve. To decide this, I considered a radius of curvature and a curve direction.

직선 또는 왼쪽 / 오른쪽 커브 인 차선 상태를 추정합니다. 이것을 결정하기 위해 나는 곡률 반경과 곡선 방향을 고려했습니다.

####Curvature 곡률

 for calculating a radius of curvature in real world, I used U.S. regulations that require a minimum lane width of 3.7 meters. And assumed the lane's length is about 30m.

실세계에서 곡률 반경을 계산할 때 최소 차선 폭 3.7m가 필요한 미국 규정을 사용했습니다. 그리고 차선 길이가 약 30m라고 가정합니다.

####Deviation

- Estimated current vehicle position by comparing image center with center of lane line.

이미지 중심을 차선 중심과 비교하여 현재 차량 위치를 추정합니다.

####Mini road map

- The small mini map visualizes above information.

작은 미니 맵은 위의 정보를 시각화합니다.

-

소스코드(파이썬) : **메인함수 main.py**

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from calibration import calib, undistort
from threshold import gradient_combine, hls_combine, comb_result
from finding_lines import Line, warp_image, find_LR_lines, draw_lane, print_road_status,
print_road_map
from skimage import exposure
input_type = 'video' #'video' # 'image'
input_name = 'project_video.mp4' #'test_images/straight_lines1.jpg' # 'challenge_video.mp4'

left_line = Line()
```

```
right_line = Line()

th_sobelx, th_sobely, th_mag, th_dir = (35, 100), (30, 255), (30, 255), (0.7, 1.3)
th_h, th_l, th_s = (10, 100), (0, 60), (85, 255)

# camera matrix & distortion coefficient
mtx, dist = calib()

if __name__ == '__main__':

    if input_type == 'image':
        img = cv2.imread(input_name)
        undist_img = undistort(img, mtx, dist)
        undist_img = cv2.resize(undist_img, None, fx=1 / 2, fy=1 / 2,
interpolation=cv2.INTER_AREA)
        rows, cols = undist_img.shape[:2]

        combined_gradient = gradient_combine(undist_img, th_sobelx, th_sobely, th_mag, th_dir)
        combined_hls = hls_combine(undist_img, th_h, th_l, th_s)
        combined_result = comb_result(combined_gradient, combined_hls)

        c_rows, c_cols = combined_result.shape[:2]
        s_LTop2, s_RTop2 = [c_cols / 2 - 24, 5], [c_cols / 2 + 24, 5]
        s_LBot2, s_RBot2 = [110, c_rows], [c_cols - 110, c_rows]

        src = np.float32([s_LBot2, s_LTop2, s_RTop2, s_RBot2])
        dst = np.float32([(170, 720), (170, 0), (550, 0), (550, 720)])

        warp_img, M, Minv = warp_image(combined_result, src, dst, (720, 720))
        searching_img = find_LR_lines(warp_img, left_line, right_line)
        w_comb_result, w_color_result = draw_lane(searching_img, left_line, right_line)

        # Drawing the lines back down onto the road
        color_result = cv2.warpPerspective(w_color_result, Minv, (c_cols, c_rows))
        comb_result = np.zeros_like(undist_img)
        comb_result[220:rows - 12, 0:cols] = color_result

        # Combine the result with the original image
        result = cv2.addWeighted(undist_img, 1, comb_result, 0.3, 0)
        cv2.imshow('result',result)

        cv2.waitKey(0)

    elif input_type == 'video':
        cap = cv2.VideoCapture(input_name)
        while (cap.isOpened()):
            _, frame = cap.read()

            # Correcting for Distortion
            undist_img = undistort(frame, mtx, dist)
            # resize video
            undist_img = cv2.resize(undist_img, None, fx=1 / 2, fy=1 / 2,
interpolation=cv2.INTER_AREA)
            rows, cols = undist_img.shape[:2]

            combined_gradient = gradient_combine(undist_img, th_sobelx, th_sobely, th_mag,
th_dir)
            #cv2.imshow('gradient combined image', combined_gradient)

            combined_hls = hls_combine(undist_img, th_h, th_l, th_s)
            #cv2.imshow('HLS combined image', combined_hls)

            combined_result = comb_result(combined_gradient, combined_hls)

            c_rows, c_cols = combined_result.shape[:2]
            s_LTop2, s_RTop2 = [c_cols / 2 - 24, 5], [c_cols / 2 + 24, 5]
```

```
s_LBot2, s_RBot2 = [110, c_rows], [c_cols - 110, c_rows]

src = np.float32([s_LBot2, s_LTop2, s_RTop2, s_RBot2])
dst = np.float32([(170, 720), (170, 0), (550, 0), (550, 720)])

warp_img, M, Minv = warp_image(combined_result, src, dst, (720, 720))
#cv2.imshow('warp', warp_img)

searching_img = find_LR_lines(warp_img, left_line, right_line)
#cv2.imshow('LR searching', searching_img)

w_comb_result, w_color_result = draw_lane(searching_img, left_line, right_line)
cv2.imshow('w_comb_result', w_comb_result)

# Drawing the lines back down onto the road
color_result = cv2.warpPerspective(w_color_result, Minv, (c_cols, c_rows))
lane_color = np.zeros_like(undist_img)
lane_color[220:rows - 12, 0:cols] = color_result

# Combine the result with the original image
result = cv2.addWeighted(undist_img, 1, lane_color, 0.3, 0)
#cv2.imshow('result', result.astype(np.uint8))

info, info2 = np.zeros_like(result),  np.zeros_like(result)
info[5:110, 5:190] = (255, 255, 255)
info2[5:110, cols-111:cols-6] = (255, 255, 255)
info = cv2.addWeighted(result, 1, info, 0.2, 0)
info2 = cv2.addWeighted(info, 1, info2, 0.2, 0)
road_map = print_road_map(w_color_result, left_line, right_line)
info2[10:105, cols-106:cols-11] = road_map
info2 = print_road_status(info2, left_line, right_line)
cv2.imshow('road info', info2)
```

```
        # out.write(frame)
        if cv2.waitKey(1) & 0xFF == ord('s'):
            cv2.waitKey(0)
        #if cv2.waitKey(1) & 0xFF == ord('r'):
        #    cv2.imwrite('check1.jpg', undist_img)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

##Result

Project Video (Click for full HD video)

Challenge Video (Click for full HD video)

##Reflection

I gave my best effort to succeed in challenge video. It wasn't easy. I have to change most of the parameters of project video. It means that the parameters strongly influenced by road status(bright or dark) or weather.
To keep the deadline, I didn't try harder challenge video yet. It looks really hard but It could be a great challenge to me.