Phase1_Advanced_Lane_Detection

April 19, 2022

1 Phase one: Lane Detection

In this first phase, the goal is to write a software pipeline to identify the lane boundaries in a video from a front-facing camera on a car. it's required to find and track the lane lines and the position of the car from the center of the lane. As a bonus, track the radius of curvature of the road too.

Assume the camera is mounted at the center of the car, such that the lane center is the midpoint at the bottom of the image between the two lines you've detected.

The offset of the lane center from the center of the image (converted from pixels to meters) is your distance from the center of the lane.

2 Imports

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from moviepy.editor import VideoFileClip
from IPython.display import HTML
from base64 import b64encode
```

3 Line Class

[ ]:
```
class LaneLines:
    def __init__(self):
        self.left_fit = None
        self.right_fit = None
        self.binary = None
        self.nonzero = None
        self.nonzerox = None
        self.nonzeroy = None
        self.clear_visibility = True
        self.dir = []
```

```
        # HYPERPARAMETERS
        # Number of sliding windows
        self.nwindows = 9
        # Width of the the windows +/- margin
        self.margin = 100
        # Mininum number of pixels found to recenter window
        self.minpix = 50

    def forward(self, img):
        self.extract_features(img)
        return self.fit_poly(img)

    def fit_poly(self, img):
        out = np.dstack((img, img, img))
        leftx, lefty, rightx, righty, out_img = self.find_lane_pixels(img)

        if len(lefty) > 1500:
            self.left_fit = np.polyfit(lefty, leftx, 2)
        if len(righty) > 1500:
            self.right_fit = np.polyfit(righty, rightx, 2)

        # Generate x and y values for plotting
        maxy = img.shape[0] - 1
        miny = img.shape[0] // 3
        if len(lefty):
            maxy = max(maxy, np.max(lefty))
            miny = min(miny, np.min(lefty))

        if len(righty):
            maxy = max(maxy, np.max(righty))
            miny = min(miny, np.min(righty))

        ploty = np.linspace(miny, maxy, img.shape[0])

        left_fitx = self.left_fit[0]*ploty**2 + self.left_fit[1]*ploty + self.
↪left_fit[2]
        right_fitx = self.right_fit[0]*ploty**2 + self.right_fit[1]*ploty +␣
↪self.right_fit[2]

        # Visualization
        c = 0
        for i, y in enumerate(ploty):
            c = c+1
            if(c == 2):
                yo = int(y)
                lo = int(l)
                ro = int(r)
```

```
            y = int(ploty[i])
            l = int(left_fitx[i])
            r = int(right_fitx[i])
            cv2.line(out, (l, y), (r, y), (0, 255, 0),20)
            if(c == 100):
                c = 0
                cv2.line(out,(lo,yo),(l,y), (255,0,0), 50)
                cv2.line(out,(ro,yo),(r,y), (255,0,0), 50)

        return out, out_img

    def extract_features(self, img):
        self.img = img
        # Height of of windows - based on nwindows and image shape
        self.window_height = np.int(img.shape[0]//self.nwindows)

        # Identify the x and y positions of all nonzero pixel in the image
        self.nonzero = img.nonzero()

        self.nonzerox = np.array(self.nonzero[1])
        self.nonzeroy = np.array(self.nonzero[0])

    def find_lane_pixels(self, img):
        assert(len(img.shape) == 2)

        # Create an output image to draw on and visualize the result
        out_img = np.dstack((img, img, img))

        bottom_half = img[img.shape[0]//2:,:]
        histogram =  np.sum(bottom_half, axis=0)

        midpoint = histogram.shape[0]//2
        leftx_base = np.argmax(histogram[:midpoint])
        rightx_base = np.argmax(histogram[midpoint:]) + midpoint

        # Current position to be update later for each window in nwindows
        leftx_current = leftx_base
        rightx_current = rightx_base
        y_current = img.shape[0] + self.window_height//2

        # Create empty lists to reveice left and right lane pixel
        leftx, lefty, rightx, righty = [], [], [], []

        # Step through the windows one by one
        for window in range(self.nwindows):
            # Identify window boundaries in x and y (and right and left)
            win_y_low = img.shape[0] - (window+1)*self.window_height
```

```
            win_y_high = img.shape[0] - window*self.window_height
            win_xleft_low = leftx_current - self.margin
            win_xleft_high = leftx_current +self. margin
            win_xright_low = rightx_current - self.margin
            win_xright_high = rightx_current + self.margin

            # Draw the windows on the visualization image
            cv2.
↪rectangle(out_img,(win_xleft_low,win_y_low),(win_xleft_high,win_y_high),
            (0,255,0), 2)
            cv2.
↪rectangle(out_img,(win_xright_low,win_y_low),(win_xright_high,win_y_high),
            (0,255,0), 2)

            y_current -= self.window_height
            center_left = (leftx_current, y_current)
            center_right = (rightx_current, y_current)

            good_left_x, good_left_y = self.pixels_in_window(center_left, self.
↪margin, self.window_height)
            good_right_x, good_right_y = self.pixels_in_window(center_right,␣
↪self.margin, self.window_height)

            # Append these indices to the lists
            leftx.extend(good_left_x)
            lefty.extend(good_left_y)
            rightx.extend(good_right_x)
            righty.extend(good_right_y)

            if len(good_left_x) > self.minpix:
                leftx_current = np.int32(np.mean(good_left_x))
            if len(good_right_x) > self.minpix:
                rightx_current = np.int32(np.mean(good_right_x))

        return leftx, lefty, rightx, righty, out_img

    def pixels_in_window(self, center, margin, height):
        topleft = (center[0]-margin, center[1]-height//2)
        bottomright = (center[0]+margin, center[1]+height//2)

        condx = (topleft[0] <= self.nonzerox) & (self.nonzerox <=␣
↪bottomright[0])
        condy = (topleft[1] <= self.nonzeroy) & (self.nonzeroy <=␣
↪bottomright[1])
        return self.nonzerox[condx&condy], self.nonzeroy[condx&condy]
```

```
    def measure_curvature(self):
        ym = 30/720
        xm = 3.7/700

        left_fit = self.left_fit.copy()
        right_fit = self.right_fit.copy()
        y_eval = 700 * ym

        # Compute R_curve (radius of curvature)
        left_curveR =  ((1 + (2*left_fit[0] *y_eval + left_fit[1])**2)**1.5)  /␣
↪np.absolute(2*left_fit[0])
        right_curveR = ((1 + (2*right_fit[0]*y_eval + right_fit[1])**2)**1.5) /␣
↪np.absolute(2*right_fit[0])

        xl = np.dot(self.left_fit, [700**2, 700, 1])
        xr = np.dot(self.right_fit, [700**2, 700, 1])
        pos = (1280//2 - (xl+xr)//2)*xm
        return left_curveR, right_curveR, pos
```

4 Prespective Transform Class

[ ]:
```
class PerspectiveTransformation:
    def __init__(self):
        """Init PerspectiveTransformation."""
        self.src = np.float32([(550, 460),     # top-left
                               (150, 720),     # bottom-left
                               (1200, 720),    # bottom-right
                               (770, 460)])    # top-right
        self.dst = np.float32([(100, 0),
                               (100, 720),
                               (1100, 720),
                               (1100, 0)])
        self.M = cv2.getPerspectiveTransform(self.src, self.dst)
        self.M_inv = cv2.getPerspectiveTransform(self.dst, self.src)

    def forward(self, img, img_size=(1280, 720), flags=cv2.INTER_LINEAR):
        return cv2.warpPerspective(img, self.M, img_size, flags=flags)

    def backward(self, img, img_size=(1280, 720), flags=cv2.INTER_LINEAR):
        return cv2.warpPerspective(img, self.M_inv, img_size, flags=flags)
```

5 Globale Var

[ ]:
```
birdeye = PerspectiveTransformation()
lanelines = LaneLines()
```

6 Blend Frames

[ ]:
```
def prepare_out_blend_frame(blend_on_road, img_binary, img_birdeye, img_fit,␣
↪Rcurve, Lcurve, pos):
    """
    Prepare the final pretty pretty output blend, given all intermediate␣
↪pipeline images
    :param blend_on_road: color image of lane blend onto the road
    :param img_binary: thresholded binary image
    :param img_birdeye: bird's eye view of the thresholded binary image
    :param img_fit: bird's eye view with detected lane-lines highlighted
    :param Rcurve: curve of the Right Lane
    :param Lcurve: curve of the Left Lane
    :param pos: offset from the center of the lane
    :return: pretty blend with all images and stuff stitched
    """
    h, w = blend_on_road.shape[:2]

    thumb_ratio = 0.2
    thumb_h, thumb_w = int(thumb_ratio * h), int(thumb_ratio * w)

    off_x, off_y = 20, 15

    # add a gray rectangle to highlight the upper area
    mask = blend_on_road.copy()
    mask = cv2.rectangle(mask, pt1=(w-(thumb_w+off_x*2), 0), pt2=(w, h),␣
↪color=(0, 0, 0), thickness=cv2.FILLED)
    blend_on_road = cv2.addWeighted(src1=mask, alpha=0.2, src2=blend_on_road,␣
↪beta=0.8, gamma=0)

    # add thumbnail of binary image
    thumb_binary = cv2.resize(img_binary, dsize=(thumb_w, thumb_h))
    # thumb_binary = np.dstack([thumb_binary, thumb_binary, thumb_binary])
    blend_on_road[off_y:thumb_h+off_y, w-(thumb_w+off_x):w-off_x, :] =␣
↪thumb_binary

    # add thumbnail of bird's eye view
    thumb_birdeye = cv2.resize(img_birdeye, dsize=(thumb_w, thumb_h))
    thumb_birdeye = np.dstack([thumb_birdeye, thumb_birdeye, thumb_birdeye])
```

```
    blend_on_road[thumb_h+(2*off_y):(thumb_h*2)+(2*off_y), w-(thumb_w+off_x):
↪w-off_x, :] = thumb_birdeye

    # add thumbnail of bird's eye view (lane-line highlighted)
    thumb_img_fit = cv2.resize(img_fit, dsize=(thumb_w, thumb_h))
    # thumb_img_fit = np.dstack([thumb_img_fit, thumb_img_fit, thumb_img_fit])
    blend_on_road[(thumb_h*2)+(3*off_y):(thumb_h*3)+(3*off_y),␣
↪w-(thumb_w+off_x):w-off_x, :] = thumb_img_fit

    # add text (curvature and offset info) on the upper right of the blend
    mean_curvature_meter = np.mean([Lcurve, Rcurve])
    # print(mean_curvature_meter)
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(blend_on_road, 'Curvature radius: ', (w - thumb_w - off_x,␣
↪(thumb_h*3)+(4*off_y)+20), font, 0.9, (255, 255, 255), 2, cv2.LINE_AA)
    cv2.putText(blend_on_road, '   {:.02f}m'.format(mean_curvature_meter),␣
↪(w-thumb_w-off_x, (thumb_h*3)+(4*off_y)+70), font, 0.9, (255, 255, 255), 2,␣
↪cv2.LINE_AA)

    cv2.putText(blend_on_road, 'Offset from center: ', (w-(thumb_w+off_x),␣
↪(thumb_h*3)+(4*off_y)+120), font, 0.9, (255, 255, 255), 2, cv2.LINE_AA)
    cv2.putText(blend_on_road, '   {:.02f}m'.format(pos), (w-(thumb_w+off_x),␣
↪(thumb_h*3)+(4*off_y)+170), font, 0.9, (255, 255, 255), 2, cv2.LINE_AA)


    return blend_on_road
```

7 Threshold methods

[ ]:
```
def threshold_rel(img, lo, hi):
    vmin = np.min(img)
    vmax = np.max(img)

    vlo = vmin + (vmax - vmin) * lo
    vhi = vmin + (vmax - vmin) * hi
    return np.uint8((img >= vlo) & (img <= vhi)) * 255

def threshold_abs(img, lo, hi):
    return np.uint8((img >= lo) & (img <= hi)) * 255
```

8 Process Frame Pipeline

[ ]:
```
def process_image(img):
    # step 1
    img1 = birdeye.forward(img)

    # step 2
    hls = cv2.cvtColor(img1, cv2.COLOR_RGB2HLS)
    hsv = cv2.cvtColor(img1, cv2.COLOR_RGB2HSV)
    h_channel = hls[:,:,0]
    l_channel = hls[:,:,1]
    s_channel = hls[:,:,2]
    v_channel = hsv[:,:,2]

    right_lane = threshold_rel(l_channel, 0.8, 1.0)
    right_lane[:,:750] = 0

    left_lane = threshold_abs(h_channel, 20, 30)
    left_lane &= threshold_rel(v_channel, 0.7, 1.0)
    left_lane[:,550:] = 0

    img2 = left_lane | right_lane

    # step 3
    img3, img4 = lanelines.forward(img2)
    Lc, Rc, pos = lanelines.measure_curvature()

    # step 4
    img5 = birdeye.backward(img3)
    out_img = cv2.addWeighted(img, 1, img5, 1, 0)

    out = prepare_out_blend_frame(out_img, img3, img2, img4, Lc,Rc, pos )

    return out
```

9 Main Code

[ ]:
```
video = ["challenge", "project"]
index = 0
clip = VideoFileClip("{}_video.mp4".format(video[index]))
out_clip = clip.fl_image(process_image)
out_clip.write_videofile("out_{}_video.mp4".format(video[index]), audio=False)
```