Element: Lane Detection

Abstract:
Traffic accidents have become one of the most serious problems in today's world. Increase in the number of vehicles, human errors towards traffic rules and the difficulty to oversee situational dangers by drivers are contributing to the majority of accidents on the road. Lane detection is an essential component for autonomous vehicles.
In this project we take a simple image/video as input data and process it to detect the lane within which the vehicle is moving. Then we find a representative line for both the left and right lane lines and render those representations back out to the video as a red overlay.

Team Members:

Name	Roll No.	Email
Aradhita Menghal	04	menghalar@rknec.edu (mailto:menghalar@rknec.edu)
Urvi Negi	16	negiu@rknec.edu (mailto:negiu@rknec.edu)
Akshat Chandak	35	chandaka@rknec.edu (mailto:chandaka@rknec.edu)
Shreyas Nemani	77	nemanism@rknec.edu (mailto:nemanism@rknec.edu)
Shubham Jha	78	jhasm@rknec.edu (mailto:jhasm@rknec.edu)

Getting Started

We developed a simple pipeline using OpenCV and Python for finding lane lines in an image, then applied this pipeline to a full video feed. We will be leveraging the popular SciPy and NumPy packages for doing scientific computations and the OpenCV package for computer vision algorithms

In [1]:

```
import numpy as np
import cv2
import tkinter
import matplotlib.pyplot as plt
import matplotlib
import os
from os.path import join, basename
from collections import deque
```

Class Line

A Line is defined from two points (x1, y1) and (x2, y2) as follows:
y - y1 = (y2 - y1) / (x2 - x1) * (x - x1)
Each line has its own slope and intercept (bias).

In [2]:

```
class Line:

    def __init__(self, x1, y1, x2, y2):

        self.x1 = int(x1)
        self.y1 = int(y1)
        self.x2 = int(x2)
        self.y2 = int(y2)

        self.slope = self.calculate_slope()
        self.bias = self.calculate_bias()

    def calculate_slope(self):
        return (self.y2 - self.y1) / (self.x2 - self.x1 + np.finfo(float).eps)

    def calculate_bias(self):
        return self.y1 - self.slope * self.x1

    def get_coordinates(self):
        return np.array([self.x1, self.y1, self.x2, self.y2])

    def set_coordinates(self, x1, y1, x2, y2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

    def draw(self, img, color=[255, 0, 0], thickness=10):
        cv2.line(img, (self.x1, self.y1), (self.x2, self.y2), color, thickness)
```

Lane Detection Pipeline

This is the entry point for lane detection pipeline. Its purpose is to assemble several steps that can be cross-validated together while setting different parameters.
It takes as input a list of frames (RGB) and returns an image (RGB) with overlaid inferred road lanes.
Eventually, len(frames)==1 in the case of a single image.

In [3]:

```
def lane_detection_pipeline(frames, solid_lines=True, temporal_smoothing=True):

    is_videoclip = len(frames) > 0

    img_height, img_width = frames[0].shape[0], frames[0].shape[1]

    lane_lines = []
    for t in range(0, len(frames)):
        resultant_lanes = get_lane_lines(color_image=frames[t], solid_lines=solid_lines)
        lane_lines.append(resultant_lanes)

    if temporal_smoothing and solid_lines:
        lane_lines = smoothen_over_time(lane_lines)
    else:
        lane_lines = lane_lines[0]

    # prepare empty mask on which lines are drawn
    line_img = np.zeros(shape=(img_height, img_width))

    # draw lanes found
    for lane in lane_lines:
        lane.draw(line_img)

    # keep only region of interest by masking
    vertices = np.array([[(50, img_height),
                          (450, 310),
                          (490, 310),
                          (img_width - 50, img_height)]],
                        dtype=np.int32)
    img_masked, _ = region_of_interest(line_img, vertices)

    # make blend on color image
    img_color = frames[-1] if is_videoclip else frames[0]
    img_blend = weighted_img(img_masked, img_color, α=0.8, β=1., λ=0.)

    return img_blend
```

This function take as input a color road frame and tries to infer the lane lines in the image.
:param color_image: input frame
:param solid_lines: if True, only selected lane lines are returned. If False, all candidate lines are returned.
:return: list of (candidate) lane lines.

These are steps of Lane Detection Pipeline

- **Grayscaling:** It converts an image to an image with only one color channel
- **Gaussian smoothing:** Smoothing is a process by which data points are averaged with their neighbors in a series, such as a time series, or image. This (usually) has the effect of blurring the sharp edges in the smoothed data. With Gaussian smoothing, the function that is used is our Gaussian curve.
- **Canny Edge Detection:**Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed.

- **Implementing a Hough Transform on Edge Detected Image:**The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure.

In [4]:

```
def get_lane_lines(color_image, solid_lines=True):

    # resize to 960 x 540
    color_image = cv2.resize(color_image, (960, 540))

    # convert to grayscale
    img_gray = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)

    # perform gaussian blur
    img_blur = cv2.GaussianBlur(img_gray, (17, 17), 0)

    # perform edge detection
    img_edge = cv2.Canny(img_blur, threshold1=50, threshold2=80)

    # perform hough transform
    detected_lines = hough_lines_detection(img=img_edge,
                                           rho=2,
                                           theta=np.pi / 180,
                                           threshold=1,
                                           min_line_len=15,
                                           max_line_gap=5)

    # convert (x1, y1, x2, y2) tuples into Lines
    detected_lines = [Line(l[0][0], l[0][1], l[0][2], l[0][3]) for l in detected_lines]

    # if 'solid_lines' infer the two lane lines
    if solid_lines:
        candidate_lines = []
        for line in detected_lines:
                # consider only lines with slope between 30 and 60 degrees
                if 0.5 <= np.abs(line.slope) <= 2:
                    candidate_lines.append(line)
        # interpolate lines candidates to find both lanes
        lane_lines = calculate_lane_from_candidates(candidate_lines, img_gray.shape)
    else:
        # if not solid_lines, just return the hough transform output
        lane_lines = detected_lines

    return lane_lines
```

Compute lines that approximate the position of both road lanes.

:param line_candidates: lines from hough transform

:param img_shape: shape of image to which hough transform was applied

:return: lines that approximate left and right lane position

In [5]:

```
def calculate_lane_from_candidates(line_candidates, img_shape):

    # separate candidate lines according to their slope
    pos_lines = [l for l in line_candidates if l.slope > 0]
    neg_lines = [l for l in line_candidates if l.slope < 0]

    # interpolate biases and slopes to compute equation of line that approximates left lane
    # median is employed to filter outliers
    neg_bias = np.median([l.bias for l in neg_lines]).astype(int)
    neg_slope = np.median([l.slope for l in neg_lines])
    x1, y1 = 0, neg_bias
    x2, y2 = -np.int32(np.round(neg_bias / neg_slope)), 0
    left_lane = Line(x1, y1, x2, y2)

    # interpolate biases and slopes to compute equation of line that approximates right lan
    # median is employed to filter outliers
    lane_right_bias = np.median([l.bias for l in pos_lines]).astype(int)
    lane_right_slope = np.median([l.slope for l in pos_lines])
    x1, y1 = 0, lane_right_bias
    x2, y2 = np.int32(np.round((img_shape[0] - lane_right_bias) / lane_right_slope)), img_s
    right_lane = Line(x1, y1, x2, y2)

    return left_lane, right_lane
```

Smooth the lane line inference over a window of frames and returns the average lines.

In [6]:

```
def smoothen_over_time(lane_lines):

    avg_line_lt = np.zeros((len(lane_lines), 4))
    avg_line_rt = np.zeros((len(lane_lines), 4))

    for t in range(0, len(lane_lines)):
        avg_line_lt[t] += lane_lines[t][0].get_coordinates()
        avg_line_rt[t] += lane_lines[t][1].get_coordinates()

    return Line(*np.mean(avg_line_lt, axis=0)), Line(*np.mean(avg_line_rt, axis=0))
```

Applies an image mask.
Only keeps the region of the image defined by the polygon formed from 'vertices'. The rest of the image is set to black.

In [7]:

```
def region_of_interest(img, vertices):

    # defining a blank mask to start with
    mask = np.zeros_like(img)

    # defining a 3 channel or 1 channel color to fill the mask with depending on the input
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    # filling pixels inside the polygon defined by "vertices" with the fill color
    cv2.fillPoly(mask, vertices, ignore_mask_color)

    # returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)

    return masked_image, mask
```

Returns resulting blend image computed as follows:
initial_img * α + img * β + λ

In [8]:

```
def weighted_img(img, initial_img, α=0.8, β=1., λ=0.):

    img = np.uint8(img)
    if len(img.shape) == 2:
        img = np.dstack((img, np.zeros_like(img), np.zeros_like(img)))

    return cv2.addWeighted(initial_img, α, img, β, λ)
```

`img` should be the output of a Canny transform.

In [9]:

```
def hough_lines_detection(img, rho, theta, threshold, min_line_len, max_line_gap):
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=min_lin
                            maxLineGap=max_line_gap)
    return lines
```

Main

In [10]:

```
if __name__ == '__main__':

    resize_height, resize_width = 540, 960
    matplotlib.use("Qt5Agg")

    plt.ion() #turn on interactive mode

    # test on images
    test_images_dir = join('data', 'test_images')
    test_images = [join(test_images_dir, name) for name in os.listdir(test_images_dir)]

    for test_img in test_images:
        figure_manager = plt.get_current_fig_manager()
        figure_manager.window.showMaximized()
        print('Processing image: {}'.format(test_img))

        out_path = join('out', 'images', basename(test_img))
        input_image = cv2.cvtColor(cv2.imread(test_img, cv2.IMREAD_COLOR), cv2.COLOR_BGR2RG
        output_image = lane_detection_pipeline([input_image], solid_lines=True)
        cv2.imwrite(out_path, cv2.cvtColor(output_image, cv2.COLOR_RGB2BGR))

        plt.imshow(output_image)
        plt.waitforbuttonpress()
    print("All images processed")
    plt.show()
    plt.close('all')

    #test on videos
    test_videos_dir = join('data', 'test_videos')
    test_videos = [join(test_videos_dir, name) for name in os.listdir(test_videos_dir)]

    for test_video in test_videos:

        print('Processing video: {}'.format(test_video))

        cap = cv2.VideoCapture(test_video)
        out = cv2.VideoWriter(join('out', 'videos', basename(test_video)),
                              fourcc=cv2.VideoWriter_fourcc(*'DIVX'),
                              fps=20.0, frameSize=(resize_width, resize_height))

        frame_buffer = deque(maxlen=10)
        while cap.isOpened():
            ret, color_frame = cap.read()
            if ret:
                color_frame = cv2.cvtColor(color_frame, cv2.COLOR_BGR2RGB)
                color_frame = cv2.resize(color_frame, (resize_width, resize_height))
                frame_buffer.append(color_frame)
                blend_frame = lane_detection_pipeline(frames=frame_buffer, solid_lines=True
                out.write(cv2.cvtColor(blend_frame, cv2.COLOR_RGB2BGR))
                cv2.imshow('blend', cv2.cvtColor(blend_frame, cv2.COLOR_RGB2BGR))
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                break
        cap.release()
        out.release()
        cv2.destroyAllWindows()
    print("All videos processed")
```

```
Processing image: data\test_images\0_gcZJDlykpYsAKmEn.jpg
Processing image: data\test_images\0_OGebOtvMuhIxH12A.jpg
Processing image: data\test_images\1200px-Strada_Provinciale_BS_510_Sebina_O
rientale.jpg
Processing image: data\test_images\1_PipigDi328-QTmnxO4Q2cQ.jpeg
Processing image: data\test_images\rohtak road.jpg
Processing image: data\test_images\Screenshot (209).png
Processing image: data\test_images\solidWhiteCurve.jpg
Processing image: data\test_images\solidWhiteRight.jpg
Processing image: data\test_images\solidYellowCurve.jpg
Processing image: data\test_images\solidYellowCurve2.jpg
Processing image: data\test_images\solidYellowLeft.jpg
Processing image: data\test_images\straight_lines2.jpg
Processing image: data\test_images\WhatsApp Image 2021-02-14 at 7.11.55 PM.j
peg
Processing image: data\test_images\WhatsApp Image 2021-02-14 at 7.29.01 PM.j
peg
Processing image: data\test_images\whiteCarLaneSwitch.jpg
All images processed
Processing video: data\test_videos\solidWhiteRight.mp4
Processing video: data\test_videos\solidYellowLeft.mp4
All videos processed
```