Team: Project 42

Report

Program to understand the movement of car from a target video

Team: **U1610041 Azizbek Kobilov**
U1610016 Akmal Karimov
U1610047 Bekhzodkhon Makhkamov
U1610052 Boburjon Bahodirov

Contents

Team Contribution

Azizbek Kobilov U1610041:

- Project planning and task distribution – As a team leader, planning workflow of project and properly distributing tasks
- Brainstorming – Researching project topic and methods of its accomplishment
- Line separation and merging – Separating lines into left and right by their slopes, finding left and right average lines and drawing new lines
- Turn prediction – Predicting turn of car according to the values of slopes of lines
- Assembling methods from each team member – Constructing the project by gathering project modules from teammates
- Optimization – Optimizing source code by removing redundant code and rewriting code for better performance

Akmal Karimov U1610016:

- Brainstorming – Researching project topic and methods of its accomplishment
- main.py – Writing workflow in a main.py
- Fixing lines – Smoothing lines by taking average of lines in the frame and from previous frame. Also, pasting frames from previous frame for continuous lines, without abruption
- FPS calculation – Calculating fps and displaying to frame
- Creating OOP approach for Video and Line objects – Collecting methods and attributes of VideoCapture object and Line object to a class representation for reusability
- Optimization – Optimizing source code by removing redundant code and rewriting code for better performance

Bekhzodkhon Makhkamov U1610047:

- Brainstorming – Researching project topic and methods of its accomplishment
- Preprocessing image – Collecting frames from video, passing for processing
- Resizing – Resizing image for each dimension
- Grayscaling – Casting image to one channel image by taking grayscale
- Applying blur function - Applying Gaussian blur method on image
- Edge detection – Detecting edges by applying Canny edge detection technique
- Optimization – Optimizing source code by removing redundant code and rewriting code for better performance

Boburjon Bahodirov U1610052:

- Brainstorming – Researching project topic and methods of its accomplishment
- Finding region of interest for each dimension – Finding 4 vertices of ROI for each given dimension of video
- Hough Lines detection – Detecting hough lines from frame by applying opencv method
- Applying mask on edge detected image and detected hough lines – Masking image with already defined ROI, in order to remove redundant parts of edge detected image. Also, in order to remove useless parts of line.
- Merging images – Merging original image with lines by using proper method from opencv

- Optimization – Optimizing source code by removing redundant code and rewriting code for better performance

Algorithm Explanation

- Video divided into frames, and each frame is passed to processing. By using object of Video class, which also uses object of VideoCapture object, read each frame of video in a while loop.
- Frame is cut in half horizontally and bottom part passed further. This is achieved by using array slicing technique. Simply, removing top half of rows in an array,
- Frame is a 3-channel image, so we convert it to one channel image by taking its grayscale. Accomplished by calling cvtColor method from opencv.
- To remove noise, we use a Gaussian blur technique to blur the image using a 3x3 kernel.
- On a blurred image, using the Canny edge detection method, passing low and high threshold 90 and 180 values respectively.
- Once edges detected, we apply a mask with ROI (region of interest). Create empty array with the shape of image, fill the polygon with given vertices of ROI. After, apply bitwise and operation on lines. So, in a result, only edges inside of that polygon are left. ROI is static, hardcoded for each 1024x768, 800x600, 640x480, 400x300 dimensions. The reason for applying the mask is to remove all redundant lines from the image and leave only lane lines.
- Frame passed to HoughLinesP method from opencv-python module; result is all hough lines on the edge detected image.
- If no lines detected, we assign lane lines from the previous frame. This fixes the case when there is no lane line detected in the video. This is achieved by saving every time lines, if they are successfully detected, as a global variables. And using them, if hough lines detection fails.
- Lines are separated into left and right, depending on their slope, if slope < 0, left lines, else right lines. All of them saved in two separate arrays.
- Finding a median of slopes and medians of intercepts (left and right separately). And based on new slope and intercept values, we find coordinates of new lines.
- Then, we try to make lines smoother by taking an average of lines from the current frame and previous. If there is no line in the current frame, this function shall fix it by replacing the absent line, with a line from the previous frame.
- Lines are drawn on an empty image with the same shape as the original frame. After, lines are masked the same way as before. This will remove ends of lines out of ROI.
- Because, original image is 3 channeled (with colors), we extend dimensions of the image with lines to 3 channels by dimensionally stacking 2 empty images.
- Finally, the image with lines and original frame (before cutting) is blended using addWeighted function from opencv module.
- Using the slope of left and right lines, it is predicted, which side car is turning. It will be easier if we use sum of two slopes. If sum is > 0.1, then car is probably turning right. Else, if sum < - 0.9, then most probably, car is turning left. Else, it is going forward.
- Processed frame is streamed to the opencv window or saved to the video file with VideoWriter.

URLs

YouTube (1024x768) - https://youtu.be/zT0dFV9nvPU

GitHub - https://github.com/Tessium/road-lane-lines-detection