ENPM673 - Perception for Autonomous Robots

Project 2

Prateek Arora, Abhinav Modi, Samer Charifa

Due date: *March 11, 2020, 11:59 p.m.*

Submission Guidelines

- The homework is to be completed in group and there should be a single submission per group.
- Your submission on ELMS/Canvas must be a zip file, following the naming convention **YourDirectoryID_proj_2.zip**. If you email ID is abc@umd.edu or abc@terpmail.umd.edu, then your DirectoryID is **abc**. Remember, this is your directory ID and **NOT** your UID. Please provide detailed instructions on how to run your code in README.md file.
- For each section of the Project, explain briefly what you did, and describe any interesting problems you encountered and/or solutions you implemented. Your report should preferable be typeset in LaTeX.
- Please submit only the python script(s) you used to compute the results, the PDF report you generate for the project and a detailed README file (refer to the directory structure below).
- Include sample outputs in the your report.
- The video outputs are to submitted as a separate link in the report itself. The link can be a youtube video or a google drive link (or any other cloud storage). Make sure that you provide proper sharing permission to access the files. **If we are not able to access the output files you will be awarded 0 for that part of the project**. On elms you can only submit one .zip file which can be of maximum 100 MB.

The file tree of your submission SHOULD resemble this:

```
YourDirectoryID_proj_2.zip
├── Code
│   ├── .py files
│   └── any subdirectories that you may have
├── Report.pdf
└── README.md
```

Dataset

1. Dataset for Problem 1
2. Dataset for Problem 2

The dataset for problem-2 has 2 different video sequences which have different camera parameters for undistorting the images and are provided separately.

Problem 1

Here, we aim to improve the quality of the video sequence provided above. This is a video recording of a highway during night. Most of the Computer Vision pipelines for lane detection or other self-driving tasks require good lighting conditions and color information for detecting good features. A lot of pre-processing is required in such scenarios where lighting conditions are poor.
Now, using the techniques taught in class your aim is to enhance the contrast and improve the visual appearance of the video sequence. You **can** use any in-built functions for the same.

Problem 2

Introduction

In this project we aim to do simple Lane Detection to mimic Lane Departure Warning systems used in Self Driving Cars. You are provided with two video sequences (both are required for this assignment), taken from a self driving car (click here to download). Your task will be to design an algorithm to detect lanes on the road, as well as estimate the road curvature to predict car turns.

Also keep in mind, that lane detection is a very popular problem and there are many implementations available on the Internet. If any part of your implementation is not designed by you, you will loose a considerable amount of points.

Suggested Pipeline

In our simplified pipeline we will process every frame independently from each other. This pipeline will rely on the fact that lanes are parallel to each other and almost always 'vertical', if you look at the image straight *from the top*. But the actual camera does not look at the road from the top, but it always looks at the road from the same angle. So if we could compute the *Homography* and distort the image so that the lanes look parallel to each other, we will get the top view.

1. Pick one image when the car moves straight and manually extract the coordinates of four points on the lanes(two for each lane). It should not matter which points you select.
2. Using these points compute the homography(you can use built-in cv2 functions for this), so that the lanes look parallel

Despite computed only on a single image, this homography transformation will work for most images, since the tilt of the camera is constant. Now, we can proceed to the main pipeline:

Step 1: Prepare the Input

1. Undistort the image (using opencv)- camera parameters are provided with the videos.
2. Denoise the image - so that the noise doesn't aggravate any of the further image processing tasks.
3. Apply edge detection to extract edges (you can use Canny or Sobel).
4. Extract the region of interest (ROI). **Note:** For this assignment, we only allow you to crop top half of the image (sky). You cannot assume that the car does not change lanes, and define small regions around lanes as ROIs - the whole bottom half of the image has to be in your ROI.

(a) Original image

(b) Transformed image

Figure 1

Step 2: Detect Lane Candidates

You could take two approaches here: use Hough Line transform (in this case, it is possible that you do not even need the homography step). Or build the histogram of lane pixels.

Hough Lines

This approach might not work well on curved roads and/or turns.

1. Find the Hough lines from the edge image acquired earlier (Fig. 2)
2. Find out the peak Hough lines and extrapolate lines in each group (Fig. 2).

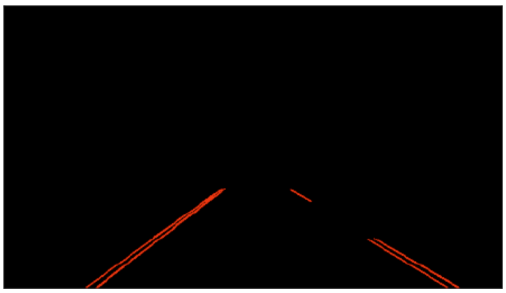
Histogram of Lane Pixels

For this approach you need the homography step. Additionally, you will need to pre-filter the image to extract candidate lane pixels; you can start with a simple Sobel filter, but eventually you might need to consider color segmentation

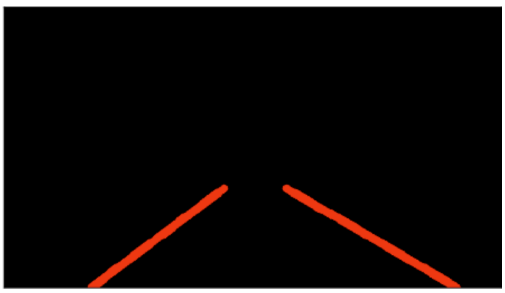
1. . Generate a histogram of pixel count along the y-axis (Fig. 3).
2. Extract the top regions with highest pixel count, they will correspond to lanes (Fig.3).

Step 3: Refine the Lane Detection

Not all lanes are straight - we recommend fitting a polynomial to the detected lane candidates for better results (Fig. 4(a)). You are required to output lane detection only for your lane. Overlay the detected lanes on the original input image, and also show the mesh covering the lane itself (an example output is shown on Fig. 4(b).



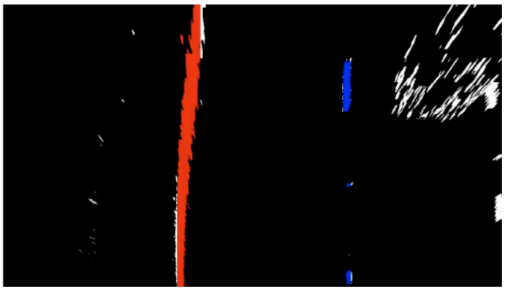
(a) Hough lines



(b) Extrapolated hough lines

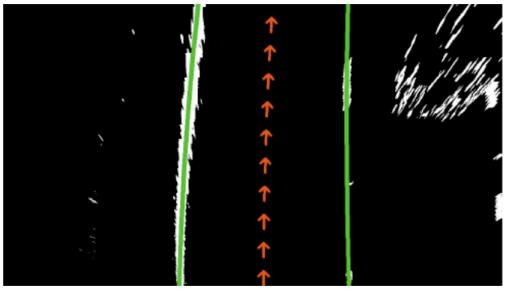
Figure 2: An approach using Hough lines

(a) Lane pixel candidates



(b) Top of the histogram (in red)

Figure 3: An approach using histograms of pixels



(a) Polynomial fitting

(b) Backprojection onto the original image

Figure 4: Polynomial fitting and an example of the final output

Step 4: Turn Prediction

One idea to predict turns is to extract the central line (averaged over left right lanes) and check if the central line has positive gradient (curving towards right) or otherwise. Another way is to use the curvature of the polynomial (fit to right or left lane) to predict the turn. There are many other ways you could explore, essentially the approach you choose should depend on the way the lane detection works.

Report

For each section of the project, explain briefly what you did, and describe any interesting problems you encountered and/or solutions you implemented. Include the following details in your writeup:

- Your understanding of homography and how it is used (if you used it).
- Your understanding of how Hough Lines work (whether you used them or not!)
- How likely you think your pipeline will generalize to other similar videos.

Please include all the results and images in the report.

Acknowledgement

We are using the Advanced Lane Detection dataset from Udacity - Self Driving Nanodegree program and KITTI dataset.
image credits: https://www.mathworks.com/help/driving/ref/segmentlanemarkerridge.html