EL 302 – DIGITAL IMAGE PROCESSING

ADVANCED LANE FINDING

Submitted by

Akshat Joshi(BT20HCS083)
Atharv Tushar Deogaonkar(BT20HCS098)
Gaytri Sran (BT20HCS190)
Vaibhav Mishra(BT20HCS202)

INTRODUCTION

Curved Lane Detection by using computer vision techniques such as perspective transform or image thresholding. Many contemporary cars come with lane assist, which, as the name implies, assists you in keeping your car in its lane.

Drivers who are drowsy or momentarily preoccupied and drift out of their lane cause many accidents on open roads.

In order to precisely position the vehicle in the center of its lane, lane assist uses a camera, often mounted in the upper portion of your windscreen. When the system notices that you are drifting from your lane, it will warn you either audibly, vibrate the steering wheel, flash a warning light, or do all three at once.

The majority of systems function best on highways, therefore it is important to note that inclement weather or low visibility might reduce their efficiency.

Nissan, Toyota, and Honda began introducing lane assist systems on a variety of passenger models in the early 2000s, but they were primarily targeted at the Japanese market. Mercedes developed the first lane assist system as we know it on their European Actros trucks in the late 1990s.

MOTIVATION

This mostly depends on the kind of trips that a driver typically takes. A lane keeping system could be seen as a guardian angel if your working life is spent pounding up and down motorways, where mile after dull mile looks pretty much the same and there is a genuine danger of falling asleep.

The price of lane assist often increases since it is bundled up with other alternatives and sold as part of a safety pack. Of course, if safety is your first priority, this is a crucial choice that lowers the likelihood of an accident occurring in the first place.

- This study forecasts how gradual adoption of vehicles with lane departure prevention (LDP) technology can reduce crashes.
- By 2020, 2.7% of single-vehicle lane departure crashes could be avoided if 8.5% of the fleet has LDP with 20% effectiveness.
- With increase in the effectiveness of LDP technology, the reduction in relevant crashes increased.
- If the LDP system is 100% effective by 2045, 66.5% of the single-vehicle lane departure crashes can be prevented.

WORK DONE

Timeline for our work has been like following:

- 1st September : Digital Image Processing Project Group was formed
- 3rd September : Topic of the Project was finalized
- 10th September : Work on Code started

- 21st September: Initial work on code completed
- 14th October: Alpha testing was done
- 18th October: Beta testing with complex Inputs
- 30th October: Error Occurred. Meetings to fix it
- 8th November : Input Error fixed
- 12th November: Work on report started
- 13th November: Simultaneous work on Presentation started
- 20th November: Project Report Completed

APPLICATION

Below are the Resources used to run the tests on our Advanced Lane Finding code:

- Movie:
    - ScreenShot: https://www.youtube.com/watch?v=f9wl35tasjw

- Image References:

    - Image 1: ./output_images/calibration1.jpg "Undistorted"
    - Image 2: ./output_images/calibration2.jpg "Road Transformed"
    - Image 3: ./output_images/bird_view.jpg "Bird View Image"
    - Image 4: ./output_images/thresholding.jpg "Thresholding"
    - Image 5: ./output_images/histogram_filtering.jpg "Fit Visual"
    - Image 6: ./output_images/result.jpg "Output"

Algorithm:

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

- Apply a distortion correction to raw images.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Camera Calibration

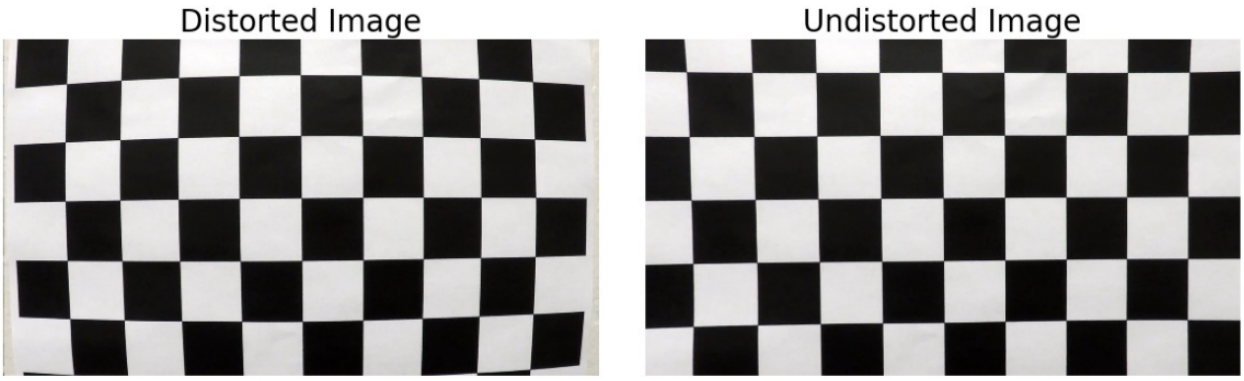
1. How to compute the camera matrix and distortion coefficients.

The code for this step is contained in the first code cell of the IPython notebook located in `./*Advanced_Lane_Finding.ipynb*` (or in lines 12 through 67 of the file called `*main.py*`).

We start by preparing "object points' ', which will be the (x, y, z) coordinates of the chessboard corners in the world. Here We are assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of coordinates, and `objpoints` will be appended with a copy of it every time We successfully detect all chessboard corners in a test image. `imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

We then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()`

function. We applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:



1. Provide an example of a distortion-corrected image.

To demonstrate this step, We will describe how We apply the distortion correction to one of the test images like this one:

2. How to perform a perspective transform

The code for my perspective transform includes a class called `Perspective_Transform`, which appears in lines 206 in the file `main.py` (in the

6th code cell of the IPython notebook). The `Perspective_Transform.transform()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. We chose the hardcode the source and destination points in the following manner:

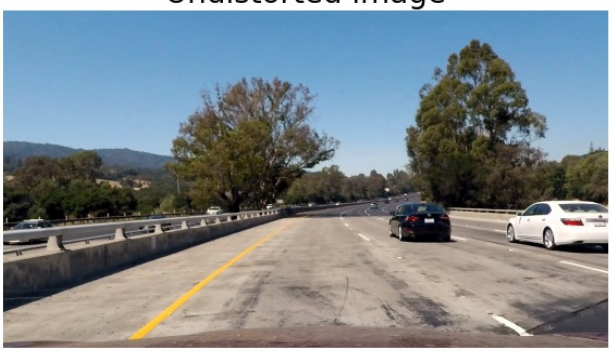
```
src = np.float32([[490, 482],[810, 482],
                  [1250, 720],[40, 720]])

dst = np.float32([[0, 0], [1280, 0],
                  [1250, 720],[40, 720]])
```

This resulted in the following source and destination points:

Source	Destination
490, 482	0, 0
810, 482	1280, 0
1250, 720	1250, 720
40, 720	40, 720

Undistorted image



Bird's view Image

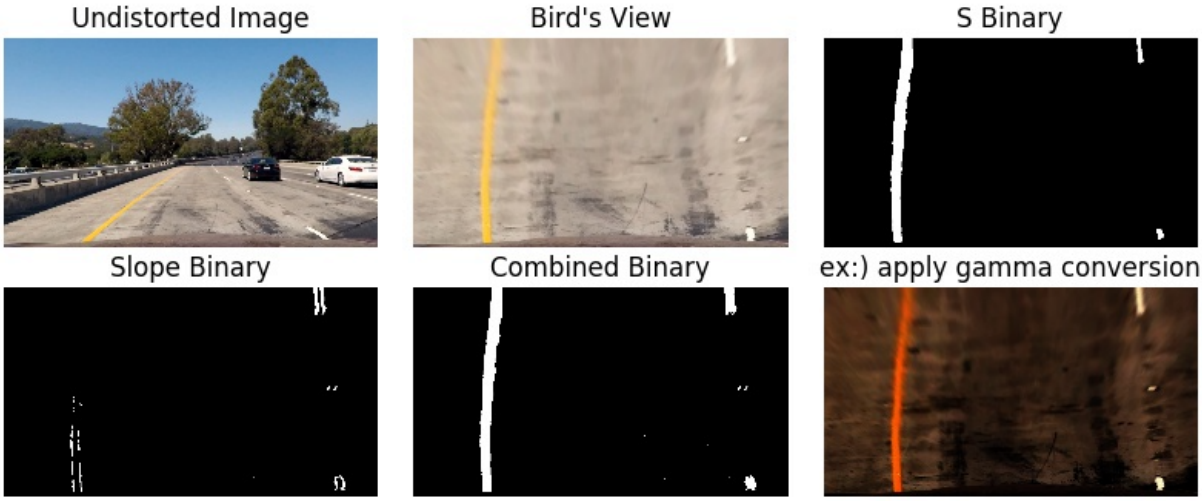
3. How to use color transforms, gradients to create a thresholded binary image

We used a combination of color and gradient thresholds to generate a binary image (thresholding steps at lines from 69 to 146 in `main.py`).
And you can see example of thresholding 16 and 17rd code cell in `Advanced_Lane_Finding.ipynb`

- The S Channel from the HLS color space, with a min threshold of 150 and a max threshold of 255, did a fairly good job of identifying both the white and yellow lane lines.

- Gradient Thresholding could get the white line. But, I have a tendency to include some noise. So We apply gaussian blur.

Here's an example of my output for this step.

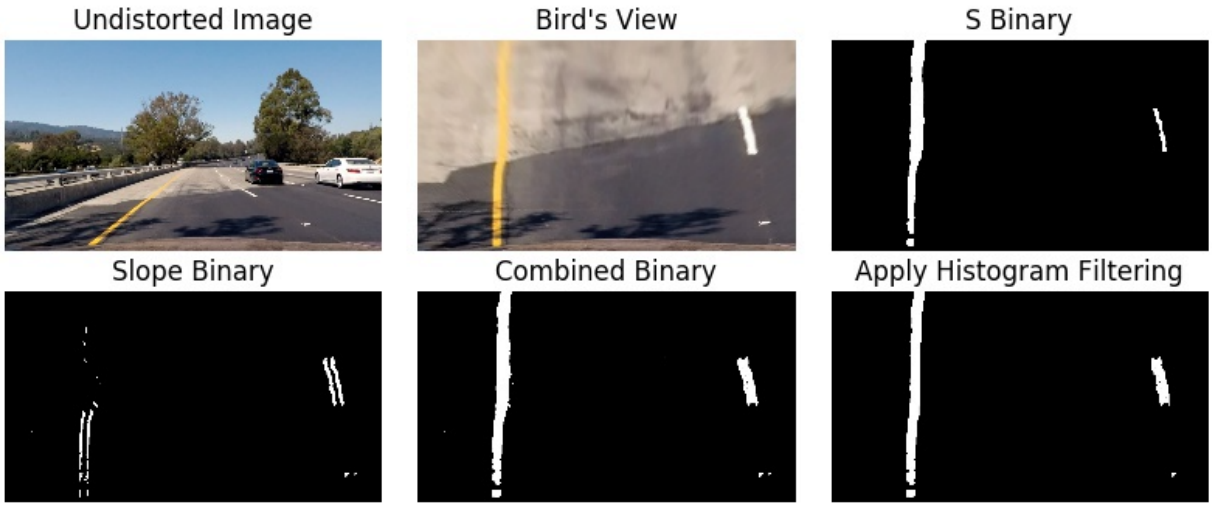


4. How to identify lane-line pixels and fit their positions with a polynomial.

- Identify peaks in a histogram of the image to determine location of lane lines.
- Mask images by peaks you get.
- Fitting a polynomial to each lane using the `cal_poly` method in lines 273 in `main.py`.

Once you've found lane lines in one frame of video, and they are actually the lines we are looking for, We can simply search within a window around the previous detection from the next frame of video.

- Search for the new line within +- 30 pixels around the old line center.
- For determining if detected lines are the real things, we check lines
- Checking that the left and right lines are roughly parallel.
- Checking that the difference between previous and current lines curvature are small.



5. How to calculate the radius of curvature of the lane and the position of the vehicle with respect to center.

- We calculate curvature by using the `__*cal_curvature*` method in lines 312 through 322 in `*main.py*`.
- Calculate the average of the x intercepts from each of the two polynomials position = **(rightx_int+leftx_int)/2**
- We calculate the position of the vehicle by using the `*add_place_to_image*` method in lines 325 through 335 in `*main.py*`.
- Calculated the distance from center by taking the absolute value of the vehicle position minus the halfway point along the horizontal axis distance_from_center = **abs(image_width/2 - position)**

Note - The distance from the center was converted from pixels to meters by multiplying the number of pixels by **3.7/700.**

6. Provide an example image of result

We implemented all pipelines in `main.py`. Main method is `process_image` in line 363 in the `Line_detector` class. Here is an example of my result in movie:

We used a mix of color and gradient thresholds for binary thresholding. However, gradient thresholding has a significant failure probability when the picture contains objects such as lines. Therefore, if more robust methods are required, simply color thresholding would be used.
And if there are vehicles in front of the cameras, we would likely be unable to accurately identify lane lines. In addition, for more effective histogram filtering, the search space might be constrained by limiting the beginning point of the line (about 100-300 for the left line) and assuming that the starting point would be restricted.

CONCLUSION

The project brings out so many learning outcomes as it is a mixture of dimensions of learning which enhances not only the presentation form but also the deep rooted process behind the working of an image processing algorithm to enhance lane assist in vehicles.

At the final stage of our idea, we've planned to give regular upgrades to our application. Once we've created and published it to our audience. This is the path to retain old visitors and to get new audiences.

The scope of improvement goes as the technology grows in this field and will definitely bring out something intelligent from the upcoming. The goal of this project is to make lane finding in vehicles more effective and efficient. Also to understand the correlation between frame by frame digital video processing and digital image processing.