Car Vision: Lane Detection and Following

by Siamak Esmi

June 2017

This is a report of a project aiming to develop an autonomous drive platform which detects road markings and keeps itself inside a lane following the course of a road. The platform hardware consists of a camera sensor, a Raspberry Pi B, an Arduino and DC-motors.

1 Introduction

The author assumes that the reader has access to the design document of software/hardware of Car Vision 2015.
Starting with a new road designation; a road composed of two solid lines defining the boundaries of the road was marked on a black mat. Beside the road borders, a broken line divides the road into two lanes. The hardware is used as is, i.e, no major modification was made. A rectangular "road" with right angles at each corner of the mat, made it possible to take advantage of the maximum area of the mat. Having the road provided with line markings and traffic signs, this project aimed to develop an autonomous car which can follow the line markings while keeping the car inside a lane and also following the traffic signs.
By providing "road data" by camera sensor, processing it and extracting the marking lines features, the DC motors pulse width is deduced.

1.1 Components of a successful approach

According to [Karouach and Ivanov, 2016], a successful development of a autonomous car requires following steps to be taken:

- **Image Pre-processing**
  - Region of interest
  - Gray scale transform
  - Binary transform
  - Blur filter
  - Inverse perspective mapping (Bird's eye view)
- **Road feature extraction**
  - Horizontal and/or vertical scan lines
  - Canny edge detection
  - Hough transform
- **road feature validation**
  - RANSAC
  - Kalman filter
  - Polynomial fitting
- **Trajectory calculation**

- Line position/incident angle
- Vanishing point

2 Car Vision 2017

The algorithms were implemented both on and off-board. It turns out that the server-client TCP-IP communication with the current hardware (wifi-dongle and router interface), i.e. transferring the image data to a backend to get processed and sending an appropriate value to DC-motors via Raspberry pi and Arduino can be expensive.

The first step taken in this project is to detect and extract road line markings. This was done after pre-processing the image data.

2.1 Image Pre-processing

Having the camera set to capture data with certain resolution, shutter speed and mode, based on the need of software, the region of interest is cropped out, transformed to a single channel image data and finally by a certain threshold criterion transformed to a binary image data.

2.2 Road feature extraction

2.2.1 ScanLines

Several algorithms were implemented in order to extract the road out of image data, namely right line, left line and the broken line.
An algorithm conceptually similar to [Vacek et al., 2007] was suggested where the recognition of road markings was made based on the length/width of the marking lines, i.e. to scan the pre-processed image data both horizontally and vertically.
This algorithm makes use of scan lines of a certain width represented in 3d-world coordination in the vehicle coordinate system. The approach is to convert the image data from perspective view (2-Dimensions) to bird's eye view (top view). As it is depicted in Figure 1, captured image data are scanned both vertically and horizontally in order to extract road features.
Bearing in mind that this implementation yields a reliable result in straight parts of the "road", on contrary, fails on curvatures. On right or left turns, the width of the detected line markings varies based on the incident angle between the camera and marking lines. The larger incident angle, the larger distance between the detected gradients of image data, thus this leads to failure in width/length ratio comparison.
Scanning, detection and classification of marking lines in this implementation is an expensive operation, as the processing time was not evaluated in the original work whatsoever.

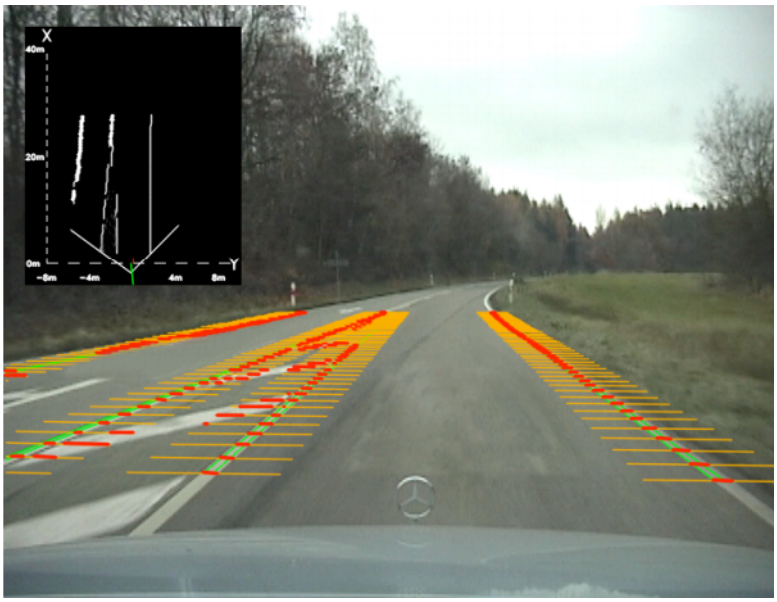

Figure 1: Illustration of vanishing point from [Vacek et al., 2007]

Different regression methods were experimented in order to extract/validate the features of the image data. K-mean regression, Polynomial fitting, Canny edge detection, Hough transform and finally minAreaRect were evaluated which the last one has the best bench-marking results in terms of robustness and time effectiveness. This method finds the convex rectangle that can cover the data in the least area.

2.2.2 Vanishing point

The next implementation was based on [Karouach and Ivanov, 2016] where the idea of the designated road was taken from. The algorithm offers an approach of discriminating marking lines according to their location in the image. Dividing the image data to 4 regions, finding the boundaries of the road is the first step. A custom scanning based on the position of the detected gradient, decreases the surfed area of the image data. Classification of lines in order of right line first, left line second and finally the broken line was instructed in the paper. Finding the first broken line and consequently seeking the second one, the intersection of classified lines was calculated and being assigned to leading point. The deviation of this leading point from the origin of the camera, determines the direction of the trajectory of the car.

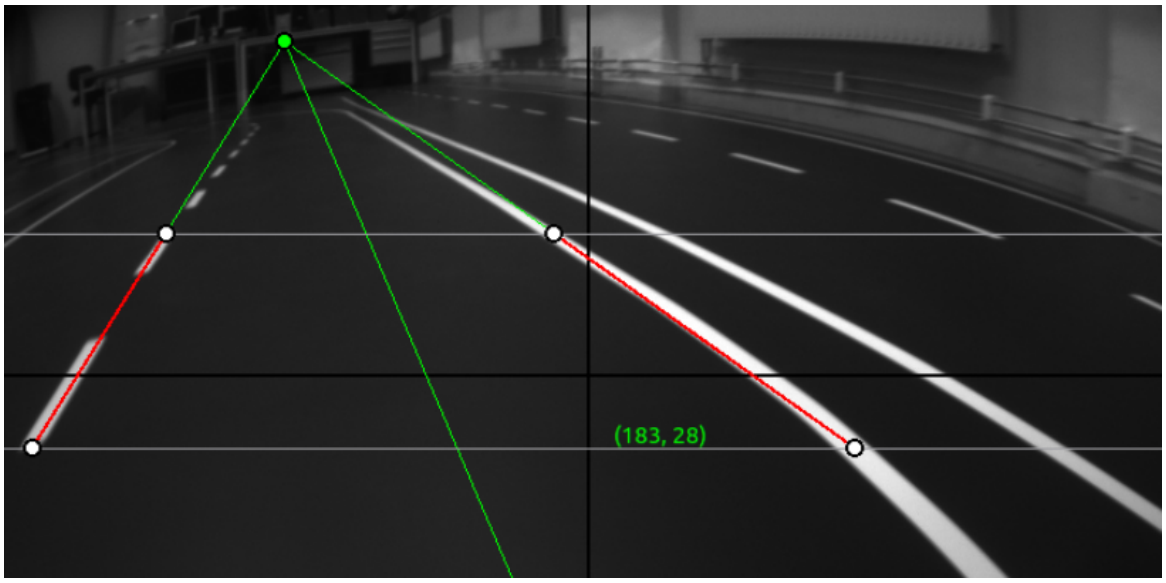

Figure 2: Illustration of detected marking lines from [Karouach and Ivanov, 2016]

2.2.3 Custom horizontal scan line

The initial idea was to design and implement a time efficient algorithm which enables the car to follow the course of a road. A sum of 30 custom horizontal scan lines in analogy with the previous work at Infotiv which deploys the first detected white pixel in the first row (from bottom) of image data, could cope with the time effective constraint of the problem. Calculating the position of right line along horizontal axis, the deviation of this value from the origin of the camera determines the direction that the car has to take.

2.3 Road feature validation

In the first implementation, classified lines were validated by vertical scan lines.
Classified lines in the second implementation were validated by checking the incident angle of the lines and the image base, i.e. bottom left of the image data (last row of the data).

2.4 Trajectory calculation

The first algorithm only extracts the marking lines and no trajectory calculation were made. In the second one, besides finding the intersection between detected lines and setting its deviation of camera origin as the error to be controlled, the incident angle of the validated right line, and finally the intersection of diagonals of the rectangle bounding the validated lines were evaluated.

2.5 Results

2.5.1 Scan lines

Figure 3: Captured image data by Pi-Camera

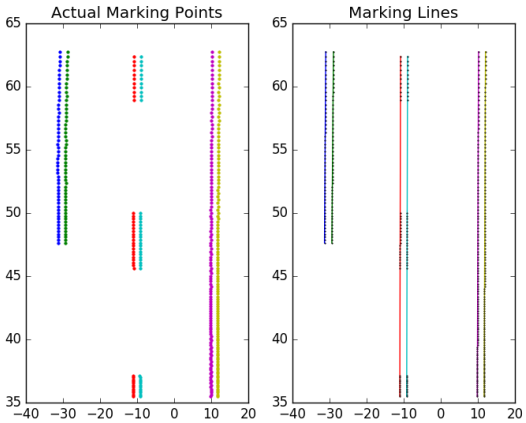

Figure 4: Top view of the image data, the length of the converted area is roughly 40 cm in real-world coordination

As it was mentioned before, the scan lines approach is a reliable one in terms of detecting the road markings. In left part of Figure 4, detected marking points (from image data Figure 5) that potentially belong to a marking line are clustered and by means of polynomial fitting, the marking lines are reproduced as illustrated in the right part of the figure. This method can roughly cover a distance of 40cm in the real-world coordination. Nevertheless, this algorithm fails when it comes to sharp curvature as described earlier. The run time is 600ms per image data.

2.5.2 Vanishing point

Figure 5: Captured image data by Pi-Camera

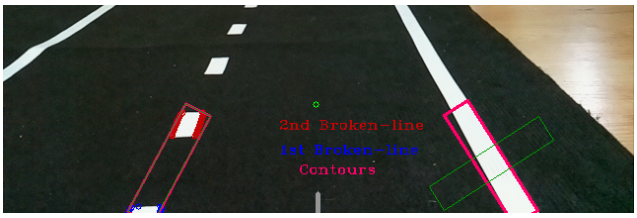

Figure 6: Validated right and broken line namely line contours are illustrated in pink, the first dashed line in blue and the second one in red

By means of a custom horizontal scan of a binary image, right line and broken lines were detected and extracted. Splitting the image data to four regions, the algorithm extracts the right line in the right bottom part. Finding the first white pixel in the last row, the scanning of the next row starts with a certain offset to the left. This customization in scanning the image data fulfill two purposes. Firstly, less amount of data being processed and secondly the method assures to high extent that the white pixel in the next row belongs to the same marking line. The first broken line should be in the left bottom part of the image data. finding the first one, the algorithm seeks for the next one in order to make an approximation of their orientation. The minAreaRect method of OpenCV returns the angle and the intersect of the diagonals of a rectangle which covers the detected marking points in a timely manner. Finding the intersect of the reproduced lines out of the angle and a point, the direction of the car is calculated. Figure 6 illustrates

the extracted road features. The camera origin is shown by a short gray line in the middle bottom of the image where the vanishing point is illustrated as a green circle.
Nevertheless, as it is depicted in Figure 7, calculation of the vanishing point location on horizontal axis needs more studies and deliberation as it exceeds the 640 which is the width of the image data. The run time per image data is 15ms.

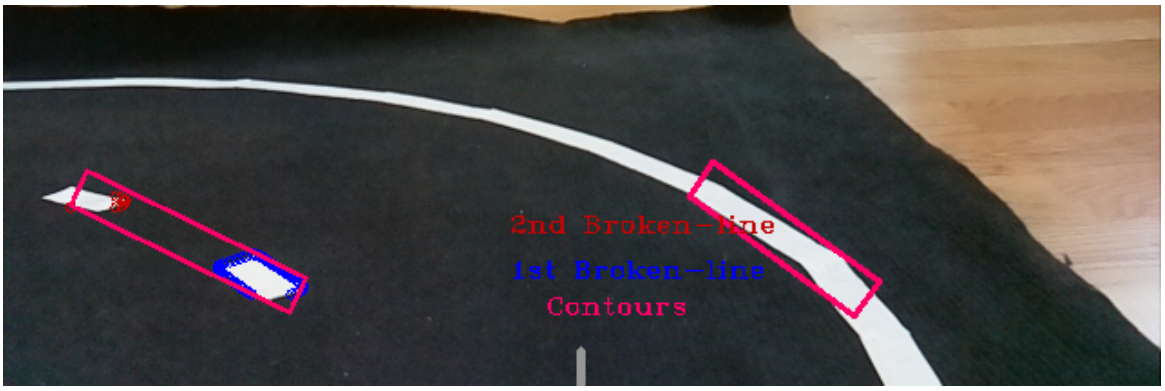

Figure 7: Illustration of the lost vanishing point

2.5.3 Custom horizontal scan lined

Calculating the vanishing point at curvatures demanded a bit of more work and as time was a determining factor, I decided to make use of functioning parts of the above algorithms and cope with the processor, DC motors and sharp curvature constraints.
Having 30 rows of the bottom right of the image data scanned, the intersection of diagonals were fed into a proportional controller to control the determine that the car has to take.

2.6 Conclusions

A complete road map from start to end is necessary to accomplish a project, where steps to be taken are clearly planned..

to be continued

2.6.1 Further improvements

The main constraints in this project are about the hardware..

to be continued

- A more realistic "road", with a more gradually curvature
- A more controllable motor and/or considering the provided power in calculation of motor pulse width
- A more powerful processor when it comes to deal with a large data (image data)
- An Ackerman-like steering as described in [Karouach and Ivanov, 2016]

References

[Karouach and Ivanov, 2016] Karouach, I. and Ivanov, S. (2016). Lane detection and following approach in self-driving miniature vehicles.

[Vacek et al., 2007] Vacek, S., Schimmel, C., and Dillmann, R. (2007). Road-marking analysis for autonomous vehicle guidance. In *EMCR*.