Application of Self-driving Car with Computer Vision

Xinzhe Liu, Shimeng Chen, Yi Qin

1 Introduction

1.1 Advanced Lane Detection

Identifying lanes on the road is a common task performed by all human drivers to ensure their vehicles are within lane constraints when driving, so as to make sure traffic is smooth and minimize chances of collisions with other cars in nearby lanes. Similarly, it is a critical task for an autonomous vehicle to perform. It turns out that recognizing lane markings on roads is possible using well known computer vision techniques.

1.2 Pedestrian Detection and Binocular Ranging

Human detection in video plays an important role in a wide range of applications that intersect with many aspects of our lives. Sometimes we need to detect pedestrian firstly, and then do the next work such as segmentation, tracking, Re-identification, action prediction and so on. In this project, we will detect human firstly and then try to confirm the distance from our car to them by binocular vision.

2 Technical Approach

2.1 Advanced Lane Detection

To detect and draw a polygon that takes the shape of the lane the car is currently in, we build a pipeline consisting of the following steps:

Computation of camera calibration matrix and distortion coefficients from a set of chessboard images;

Distortion removal on images;

Application of color and gradient thresholds to focus on lane lines;

Production of a bird's eye view image via perspective transform;

Use of sliding windows to find hot lane line pixels;

Fitting of second degree polynomials to identify left and right lines composing the lane;

Computation of lane curvature and deviation from lane center;

Warping and drawing of lane boundaries on image as well as lane curvature information.

2.2 Pedestrian Detection and Binocular Ranging

Use HOG algorithm to do feature extraction; If it's possible, use improved LBP to reduce false positive results;

Use SVM to classify the sub-image. It will give a probability that there is a pedestrian in the image.

Use sliding windows to division frame.

Use SIFT to find the corresponding points.

Calculate the 3-D position information and then get the distance.

If it's possible, use CNN to do double check.

If it's necessary and possible, use FPGA to accelerate the arithmetic speed.

3 Milestones achieved so far

Completed part (**Pedestrian Detection**)

1. Technical detail

1.1. Get HOG feature vector

This method is based on evaluating well-normalized local histograms of image gradient orientations in a dense grid. The basic idea is that local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions, even without precise knowledge of the corresponding gradient or edge positions.

1.1.1. Gamma/Color Normalization

Square root gamma compression of each color channel improves performance. Setting γ as 0.5 is pretty good.

1.1.2. Gradient Computation

For differential operator, simple $1-D\ [-1,0,1]$ masks at $\sigma = 0$ work best. For color images, we calculate separate gradients for each color channel, and take the one with the largest norm as the pixel's gradient vector.

1.1.3. Spatial/Orientation Binning

Each pixel calculates a weighted vote for an edge orientation histogram channel based on the orientation of the gradient element centred on it, and the votes are accumulated into orientation bins over local spatial regions called cells. Cells can be either rectangular or radial (log-polar sectors). The orientation bins are evenly spaced over $0° - 180°$ ("unsigned" gradient) or $0° - 360°$ ("signed" gradient).

To reduce aliasing, votes are interpolated bilinearly between the neighboring bin centers in both orientation and position. The vote is a function of the gradient magnitude at the pixel, either the magnitude itself, its square, its square root, or a clipped form of the magnitude representing soft presence/absence of an edge at the pixel. In this project, we use the magnitude itself.

1.1.4. Normalization and Descriptor Blocks

The final descriptor is then the vector of all components of the normalized cell responses from all of the blocks in the detection window. In this project, we overlap the blocks so that each scalar cell response contributes several components to the final descriptor vector, each normalized with respect to a different block.

1.2. Get CENTRIST visual descriptor

Census Transform is a nonparametric local transform originally designed for establishing correspondence between local patches.

Census transform compares the intensity value of a pixel with its eight neighboring pixels. If the center pixel is bigger than (or equal to) one of its neighbors, a bit 1 is set in the corresponding location. Otherwise, a bit 0 is set.

The eight bits generated from intensity comparisons can be put together in any order

(we collect bits from left to right, and from top to bottom), which is consequently converted to a base-10 number in $[0,255]$. This is the Census Transform value (CT value) for this center pixel. Note that the Census Transform is equivalent (modulo a difference in bit ordering) to the local binary pattern code $\mathrm{LBP}_{8,1}$.

CENTRIST means "CENsus TRansform hISTogram".

1.3. Sliding window

In spite of its good performance, the approach of sliding window classification is often criticized as being too resource and computationally expensive. The integral image/histogram and the efficient sub window search help to alleviate the speed problem. Within the framework of the integral image/histogram, the extraction of features for scanning windows has a constant complexity O(c) (two vector addition and two vector subtraction).

1.4. Classifier

We use a soft $c = 0.01$ linear SVM trained by ourselves and Navneet Dalal respectively (slightly modified to reduce memory usage for problems with large dense descriptor vectors).

1.5. Note

The detection algorithm runs in 4 nested loops (at each pyramid layer):

```
1.  loop over the windows within the input image
2.      loop over the blocks within each window
3.          loop over the cells within each block
4.              loop over the pixels in each cell
```

As each of the loops runs over a 2-dimensional array, we could get 8 nested loops in total, which is very-very slow. To speed the things up, we do the following:

a) Loop over windows is unrolled. Inside we compute the current search window.

b) Loop over the blocks is also unrolled. Inside we use pre-computed lookup table of block Data to set up gradient and histogram pointers.

c) Loops over cells and pixels in each cell are merged (since there is no overlap between cells, each pixel in the block is processed once) and also unrolled. Inside we use lookup table of Pix Data to access the gradient values and update the histogram.

1.6. Integration

Firstly, we get the intersection of two rectangles which come from two classifiers with HOG feature extraction respectively. And then we try to know if it's possible to be a correct result. A normal human standing at a corner usually has the body ratio of 3:7 (width: height) and a normal walking human usually has the body ratio of 5:7. So check the length-width ratio is a useful method. Besides, because of the algorithm, the smallest human we can detected is about 64×128. Hence if the intersecting rectangle is much smaller than 64×128,

it's almost impossible to be right.

The C^4 detection has distinct false positive sensitive field to HOG+SVM detector. But it always mistake the position and size (too small) of box rectangles. So we use it to verify if the intersecting rectangle really correct. If the intersecting rectangle similarly contain the rectangle of C^4, we'll believe that it's correct.

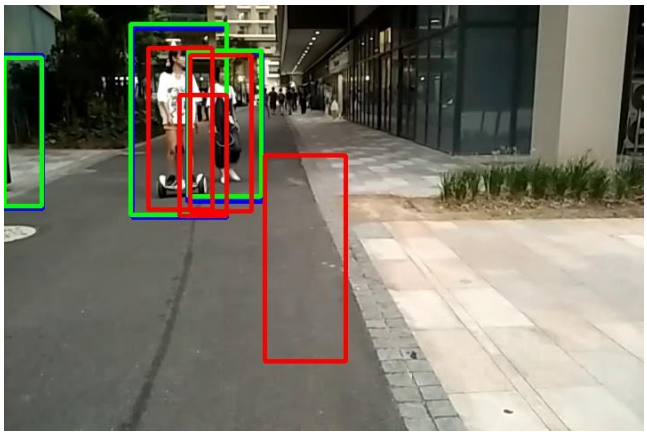

Figure 1. The results from three detector respectively. Green rectangles come from HOG+SVM trained by ourselves. Blue rectangles come from HOG+SVM trained by Navneet Dalal. And red rectangles come from C^4 detector.

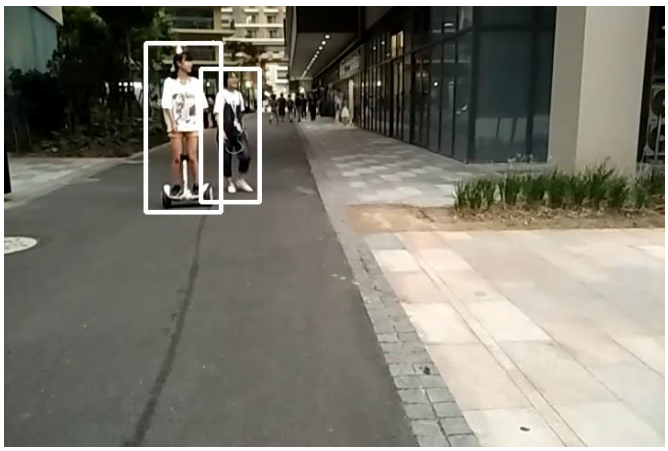

Figure 2. The final result after Integration. The size and position of rectangles are satisfactory

4 Remaining Milestones

Dec. 17th, Complete the lane detection
Dec. 24th, Complete the distance measurement and Prepare for the presentation
Dec. 30th, Finish final report