Final Project Report

-Real-time traffic signs detection

I. Problem statement.

In the real world traffic system, to make sure safety and reliability of whole traffic network, all vehicles inside it must obey local traffic rules, and in most cases, a common driver could learn these rules by reading the traffic signs, unfortunately, there could be thousands of traffic signs in a single route, so it could distract the driver and lead to some tragedy, especially when there's a complex road condition.

And in the final project of last semester, we knew how to use the Neural Network to recognize the traffic signs, with the images of candidate traffic signs. However, in the real world scenario, most of the traffic signs are surrounded by unrelated noises, like in figure 1, we need to know how to separate the no right turn sign, which is located in the center of image from the original image.

fig 1

So, generally, we'll try to solve these problems:

1. Extract traffic signs from original pictures, or frames, which are captured in real world.
2. The detecting algorithm should be robust enough to deal with slightly shape changed traffic signs, like a circle-border sign could look like ellipse when we observe it from left or right side of it.
3. To be a real-time system, the program should be able to process the data from computer cameras, which are the frames. And mark the possible traffic signs positions.

To be mentioned, since the time is limited, in this project, we will process only the traffic signs with circle or ellipse borders, like no right-turn and speed limit signs in most European and Asian countries.

fig 2

II. Proposed solution.

Generally, we could divide the procedures into 4 steps: color threshold, draw contours, corner detection(optional) and shape recognition, and input file should be color image with RGB channels. To be more specific, we list the whole procedures below, in fig 3. As we can see, the program will deal with the functions displayed in left rectangle, which is also known as Traffic Sign Detector, it could solve “where is the traffic signs”. And then the results of it will be delivered to Classifier,

which could let us know “what kind of traffic signs they are”.

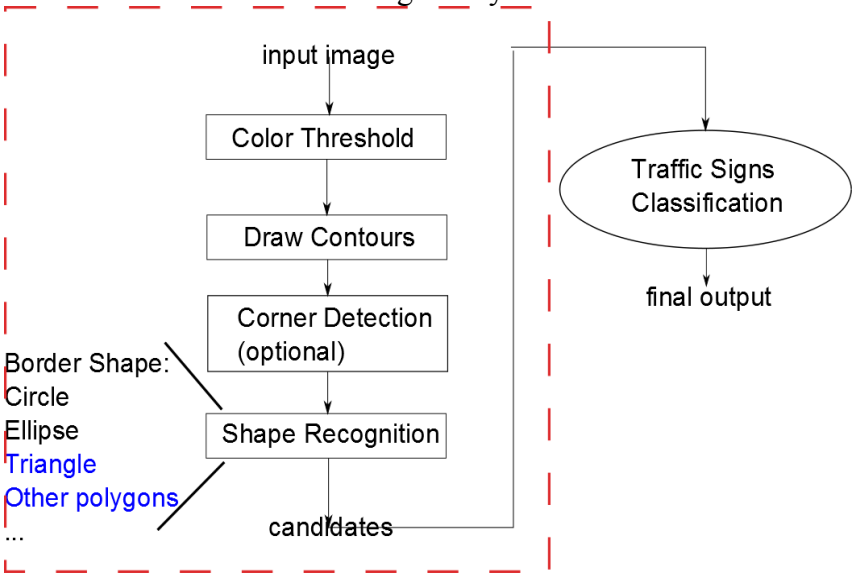

fig 3

Apparently, in this project, we'll focus on the detection function. To be start, the raw image or frame will be handled to the detector. As we know, we can not apply the corner or shape detector in raw image, since there are too many noises, it could influence the result of recognition. So, some pretreatments should be introduced, like color threshold.

As the common experiences, most of the traffic signs are painted with conspicuous color, like red, orange or yellow. So the color threshold could help us to divide this signs with other irrelevant signals. However, here comes another problem, since in common RGB metric system, the change of light might impact the threshold seriously, like in fig 4, the no right-turn sign was covered by the shadow of right building, which could make the red color look like black, and if we sample a point

fig 4

from the red border, we have the result as [49, 22, 37], in RGB metric, it's typical gray or black, but not red, no mention the dark night scenarios.

To solve this problem, we need to introduce a new color metric, which can not be impacted by light easily, like HSV, or HSL. Similarly, the HSV metric has 3 components, which are Hue, Saturation and Lightness[1], not like the RGB metric, lightness is separated, which means we could control

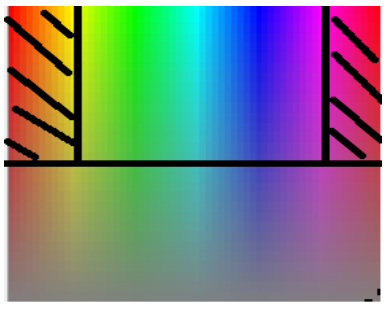

Fig 5-1

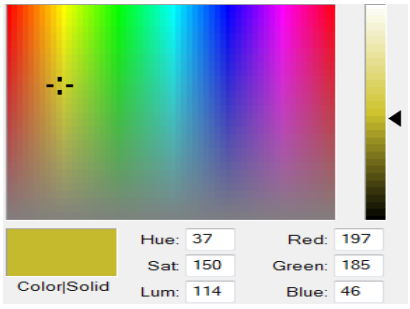

fig 5-2

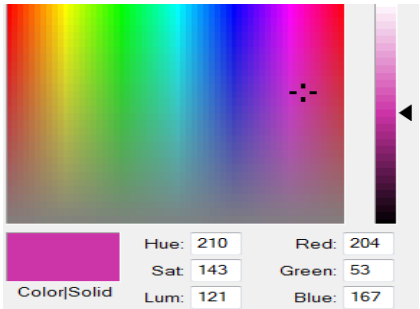

fig 5-3

the influence from light changes. Then the next step is determining the color threshold, in this case, it is red color threshold. When we observe the color spectrum, like fig 5-1, we could say that most of the available red colors are located in the sections with dark shadows, to find out the specific values, we could use the color tools, as fig 5-2 and 5-3 display. In this case, lets take [H,S,V] = [0~20, 50~255, 50~255] and [150~179, 50~255, 50~255] as our threshold, the reason why we set H=179 as the upper limit of Hue is that 179 is the cap of Hue in OpenCV HSV metric. To demonstrate the result of color threshold, let's take fig 4 as our input image, and the result of color threshold is displayed as fig 6:

fig 6

After that, we could see most of the points are separated after color threshold operation. In this case, we need to know the relationships of this points, in another words, we should know if a point belongs to a specific object, and organize the set of points as single object. To achieve this, we could use the find contour function that provided by OpenCV, the result could be like fig 7:

fig 7

As the results indicates, all points are related by objects, and the outter border of traffic sign is represented as single circle, and more, we'll filter those objects who has only few of points in next step, which could help us to improve the result of algorithm.

Continuously, due to we apply only the circle and ellipse detector in this project, we will not talk about the corner detection function. But if the border of traffic signs are triangles or any other polygons, then we should use the corner detection to identify them.

Then the critical step, shape recognition, till now, we get the intermediated image that we expected without any error. And the goal of this step is, in all the cadidates. determine which object is the traffic sign. Majorly, 2 shape detectors need to be instructed, which are circle and ellipse, as we all know, circle could be treated as special case of ellipse, so, we could focus on how to find out ellipse from all objects.

To make sure if a shape is ellipse, a couple of techniques could be helpful, like

1. if we use a rectangle to cover whole ellipse, like in fig 8, we could see that the effective

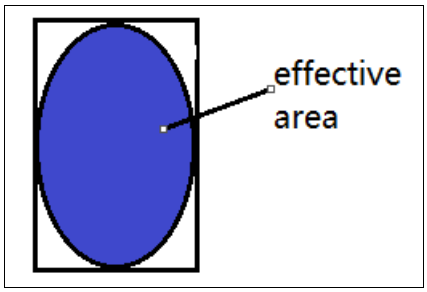

fig 8

area devides by the whole rectangle are, it should satisfy a fixed ratio, like greater than 0.5, so any shape that can not meet this requirement could be treated as non-ellipse.
2. We could also give program a range about width/height ratios, any object exceed this ratio range will be excluded from cadidates.
3. One reliable technique will be corner detection, like in figure 9, we could specify 4 masks,

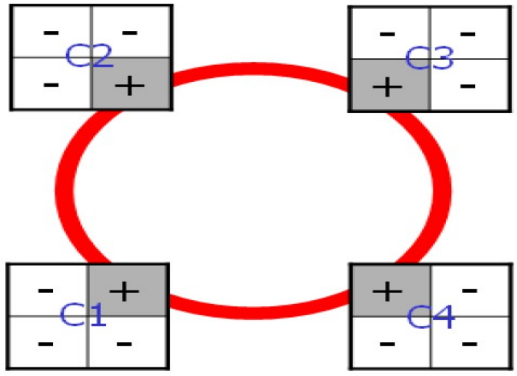

fig 9

related to 4 corners of ellipse. However, since we need to scan all possible windows in the objects, the efficient of this method could be low, which means it might not eligible for real-time detection, since it requires that the method should have short response time.

Till now, we instructed 3 kinds of possible techniques to apply ellipse detection, to improve the accuracy of our algorithm, our recommandation is applying all 3 of them, and taking the output of one technique as the input of another one, which is so called "hybrid ellipse detector".

Even that, we can not avoid the false detection, like in fig 10, which is the result of detection about

fig 10

fig 4, obviously, we have one error in this detection, since the contour of the object might be close to ellipse.

III. Implementation details.

There are some issues left in this project:

1. It seems like we can not apply color threshold and corner detection in the same time, since after color threshold, the edge of contour could be no more smoothing, so the corner detection could always return a number of corners in later computation, which could make the result to be meaning less.
2. Most of the parameters need further adjusting, since they might not be suitable for every image, like the range of red color threshold and error threshold.
3. Incorrect fitting still exists, especially in some images, which have a lot of red color blocks.

Then, for further improvement, we have several ideas:

1. Increate the robust of algorithm by intruducing new techniques, and combine it with existing methods.
2. Instead of using color threshold in whole picture, we could applying drawing contour and shape recognition first, then filter the red color in limited area, which could let the program be able to compute the threshold automatically.

IV. Results and discussion.

In order to verify our algorithm, and find out the accuracy, we could use a group of images to test our program, here's some of them:

Fig 11-1

fig 11-2

fig 11-3

And the output is:

Fig 11-4

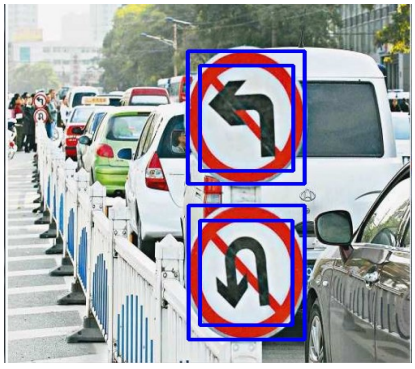

fig 11-5

fig 11-6

For more details of results, in 20 images, we have 10 images with absolute correct results, and 7 of them have both the correct and incorrect detection results, and we also have 3 incorrect results images. So, majorly, the accuracy of our algorithm is roughly 85%.

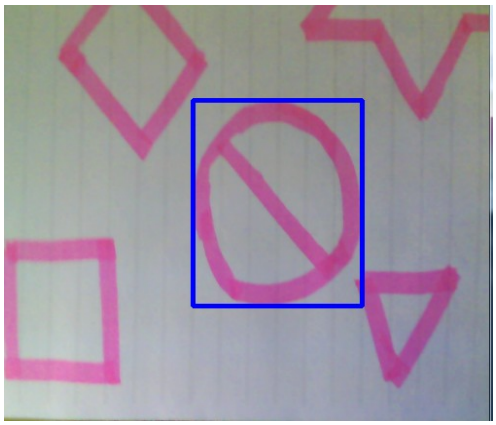

fig 12-1

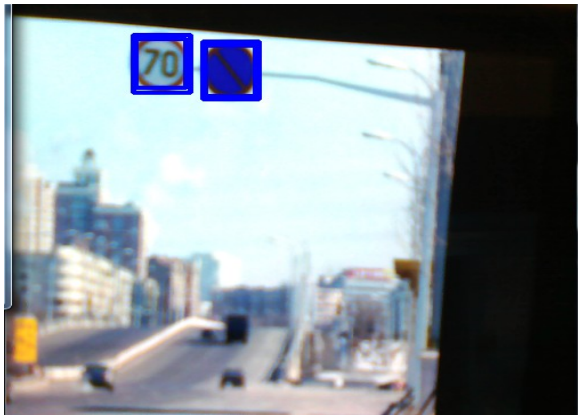

fig 12-2

Then let's take a look at the real-time situation, with the input from camera. In the first test, to make the result be meaningful, we draw some shapes in a paper, then use our camera to capture these shapes, to check if the program could identify ellipse from other shapes. The result could be confirmed by fig 12.

Then feed the program some real-world pictures, like in fig 12-2. The same as the static image mode, the camera mode could return the results based on each frame, then keep tracking the position of signs.

Reference

[1]HSL_and_HSV, Wikipedia, http://en.wikipedia.org/wiki/HSL_and_HSV

[2]Real-time Traffic Sign Detection, Hassan Shojania, shojania@ieee.org

[3]Traffic Sign Recognition with Multi-Scale Convolutional Networks, Pierre Sermanet and Yann LeCun, Courant Institute of Mathematical Sciences, New York University