COMP 4102 Traffic Sign Recognition

Final Project Report

Team members:

Jiaming Mei 101014538
Yu Tang 101130194

Abstract:
Traffic Sign Recognition is to recognize and identify the traffic signs in Canada. The project needs to deal with image processing (shape and color processing) and image recognition (text and symbol recognition). The project delivers two segmentation programs that find the road sign in the image by colors and contours. After segments, a matching program is performed to compute the similarity with the dataset and present final recognition result.

Introduction:
Traffic signs recognition is mostly for smart driving and manless driving in the field of artificial intelligence and computer vision. The recognition should consider images sent from the frontier camera of transporting tools, perform the recognition and return the correct meaning of signs to the driver or user. In the program, images of signs should be provided and a dataset should be included to perform the computation. OpenCV is the primary use to achieve image reconstruction, image segmentation, and shape recognition in the traffic sign recognition system. The program is implemented in Python with OpenCV libraries, and the dataset is provided.

Background:
In the paper “Hardware/Software Co-Design of a Traffic Sign Recognition System Using Zynq FPGAs”, we know that Sign detection is primarily finding the region of interest (ROI) that may contain signs. One of the most popular methods is color-based segmentation. In the project, we decided to use the color detection method to segment the traffic sign from images. To deal with image recognition, the project uses the point matching method, which is to calculate the probability of matching points of the image with the traffic sign database to get the optimal matching sign.

Approach:

Segmentation:
The first thing we do is that we want to find where the sign is located on the image. We would like to locate it, and crop the original image to the little image that only contains the sign itself. This is done by two approaches of segmentation - segmenting by colors, and segmenting by contours.

Color segmentation:

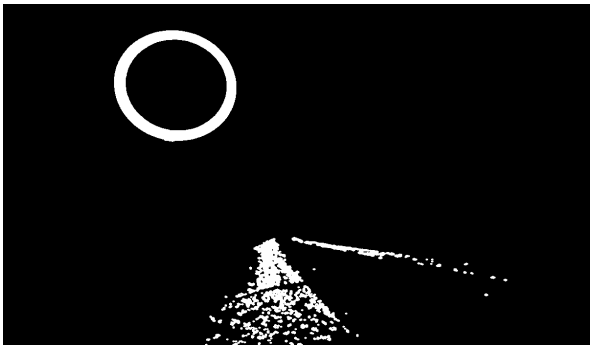
The first approach is using colors to locate the approximate location of the sign. After investigating the variable sets of Canadian traffic signs, we can conclude that there are mainly four different colors of signs in Canada - red, blue, white, and yellow. In this case, we only need to consider these four colors and proceed the image with all color filters. We are using opencv to filter colors in range in HSV value and hence we have a table of the corresponding HSV values that are used to compute the mask.

```
color_filter = dict()
color_filter["red"] = [([0,30,100], [10,255,255]), ([170,30,100], [180,255,255])]
color_filter["yellow"] = [([22, 93, 0], [45, 255, 255])]
color_filter["blue"] = [([60, 35, 140], [180, 255, 255])]
```

With color segmentation, the precision is high but the accuracy of the prefered area is low since there can be anything in the image with the same color of the signs. In this case, the color segmentation is mainly used for narrowing the area to perform the contour segmentation.

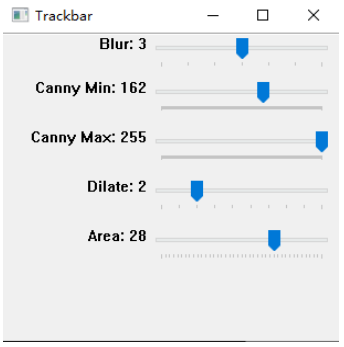
We can see that with an image that is clear of colors, color segmentation is very accurate in separating the sign and the background. However, when it comes to the background of complicated element composition, we can see the color filter can be obtaining too much content from the image. In this case, after applying the color segmentation, we also need a segmentation called the contour method.

Contour Segmentation:
Contour segmentation works by finding all contours in an image. A contour is defined as the connected or approximately continuous edges that wrap a specific shape. It is also widely used in shape detection where it approximates the shape by the number of arclength and “straight line” in the contour. However, in this project, the contour is found on the mask from the color segmentation.



As an example, the mask (right) from the color segmentation from the figure has a very obvious shape of a circle, and several noise pixels on the sides. With contour segmentation, we will be able to find the contours of the obvious shape, and the noise pixels will not be counted as a proper “shape area”.

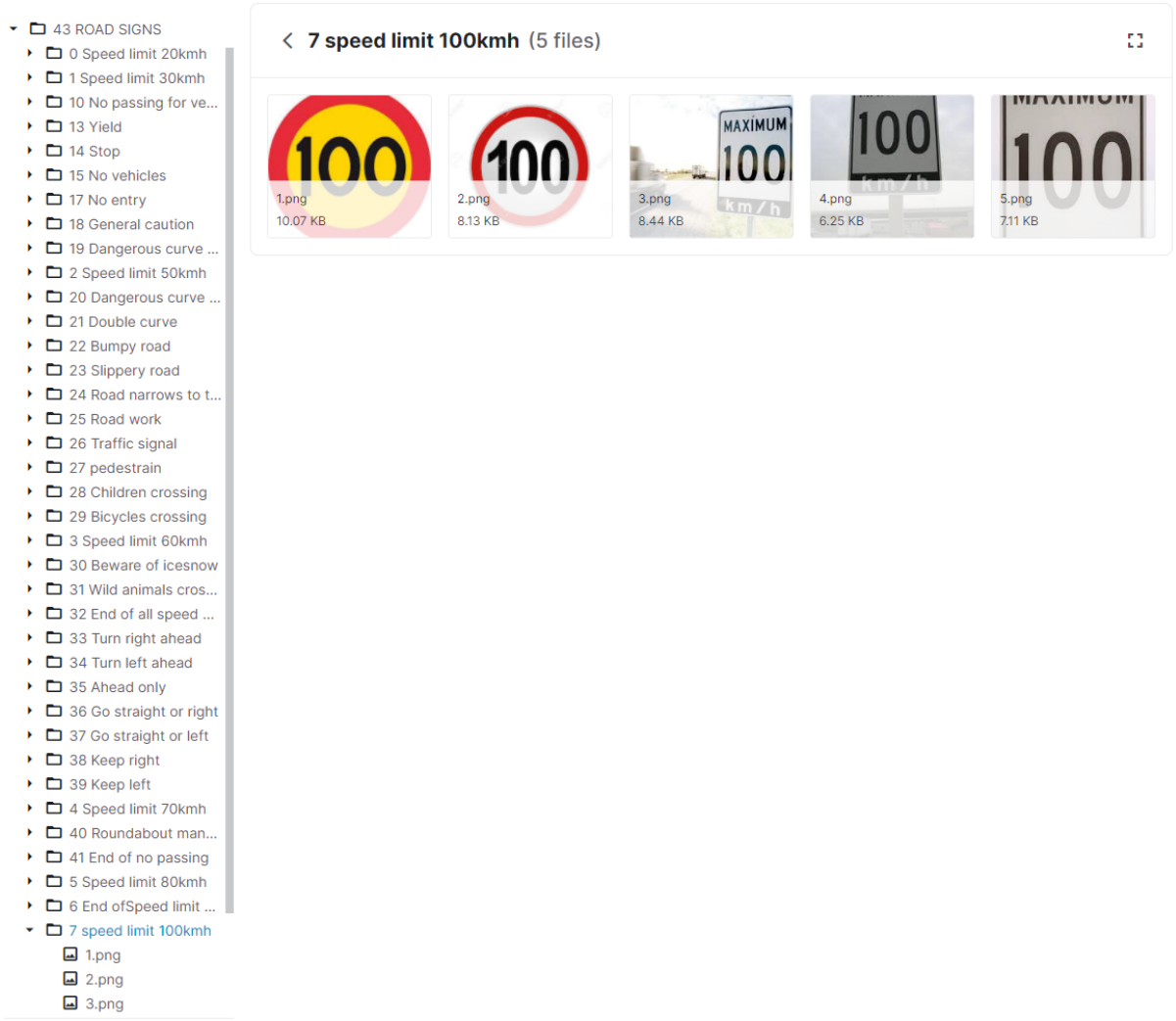
The findContours function from OpenCV relies on the Canny Edge detector, where it finds edges with the minimum and maximum thresholds and contour and we may also need variables for the matrix size of Gaussian Filter and the size of dilation. In this case, we can set up a trackbar window that dynamically changes the value of those thresholds by the user to obtain the most approximate contour.



From this example we can see the magic of dynamically threshold altering. The user is able to set the blur value, Canny thresholds, the ratio of dilation and finally, the minimum area of the contours to get rid of small noises pixels. After all those operations, with the two steps segmentation, we are able to crop the original image to the area of where the signs are located by the contours bounds.

Recognition

Keypoint matching:
The next stage with the segmented single image of signs is the recognition, where the computer recognizes the meaning of the signs by comparing them to the datasets that we prepare. The dataset contains all types of Canadian road signs with multiple samples per sign. The recognition is to compare the segmented image with all traffic signs samples to calculate the highest probability of correspondence; which is the number of matching keypoints per sample.



We learn feature extraction from images features and homography. It is known that for every corner from the Canny Edge detection, there can be a keypoint for the image. And by comparing the key points from two images, we can find the matching keypoints with a specific threshold. Where we use BFMatcher to generate matching keypoints in order of similarity, we use the SIFT algorithm as the keypoint detector. (Note that SIFT can be paid for commercial usage). By finding the maximum numbers of matching keypoints from the segmented sign image and all road signs samples, we conclude the recognition of the sign.

Results:

Color Segmentation Samples:

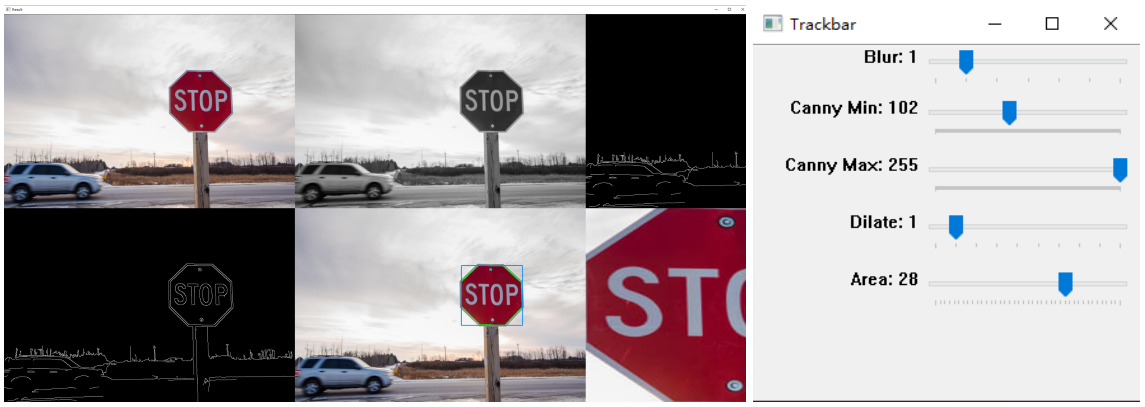
(Figure: No passing sign and No passing sign in BLUE mask)

(Figure: No passing sign and No passing sign in RED mask)

(Figure: No passing sign and No passing sign in YELLOW mask)

Contour Segmentation Sample:

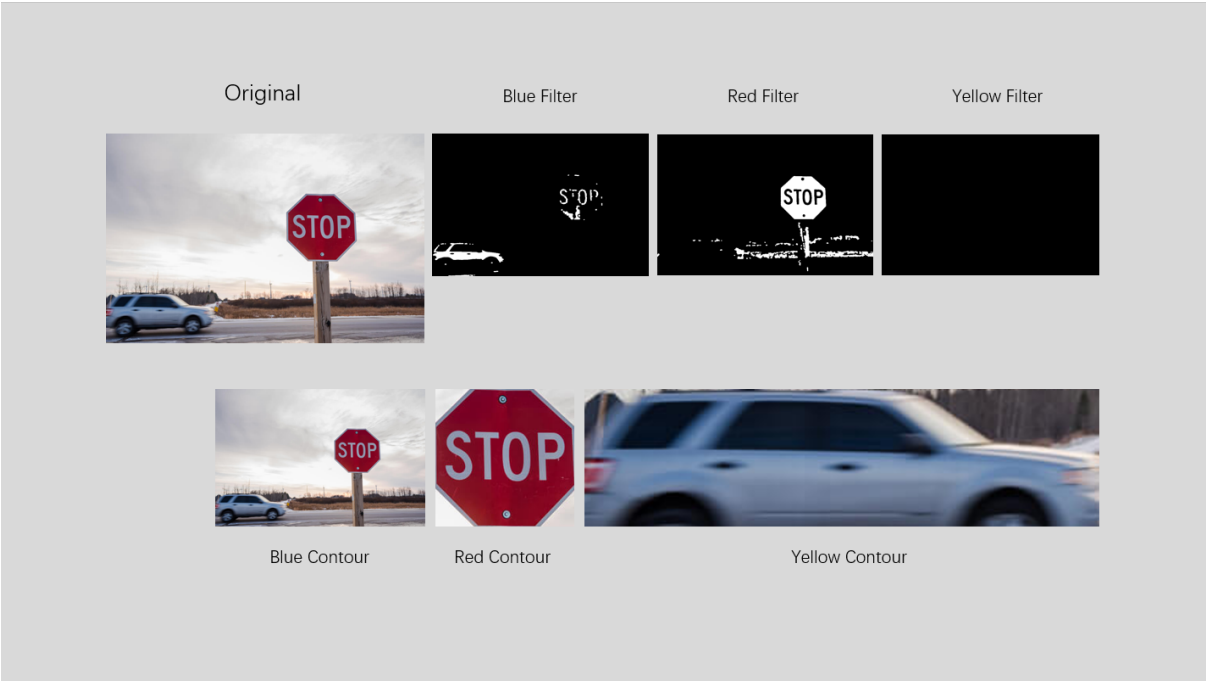
(Figure: No passing sign after contour segmentation by BLUE, RED, YELLOW mask)

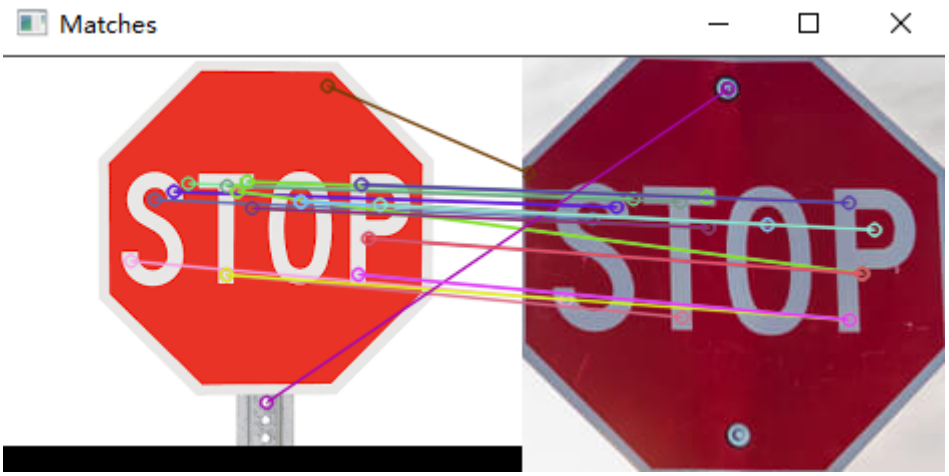


(Figure: Procedure of contour dynamic segmentation)

Recognition Result:

1. Stop sign






```
Computing  stop2_blue.png :
stop2_blue.png :
[('34 Turn left ahead', 50), ('5 Speed limit 80kmh', 49), ('1 Speed limit 30kmh', 45)]

Computing  stop2_red.png :
stop2_red.png :
[('14 Stop', 48), ('14 Stop', 27), ('1 Speed limit 30kmh', 25)]

Computing  stop2_yellow.png :
stop2_yellow.png :
[('14 Stop', 30), ('14 Stop', 23), ('14 Stop', 23)]
```

Wrong approach with BLUE filter. Correct Approach in RED and YELLOW filter.

The overall accuracy shoots at around 60%. It performs not as good when comparing speed limitation signs (text recognition required) and approaches good on signs with single color -- Stop sign, Yield sign, No passing, etc.)

More tests:


```
Computing  yield-signs-29587-1g_blue.png :
yield-signs-29587-1g_blue.png :
[('13 Yield', 42), ('13 Yield', 41), ('25 Road work', 36)]

Computing  yield-signs-29587-1g_red.png :
yield-signs-29587-1g_red.png :
[('13 Yield', 43), ('13 Yield', 41), ('27 pedestrain', 40)]

Computing  yield-signs-29587-1g_yellow.png :
yield-signs-29587-1g_yellow.png :
[('13 Yield', 42), ('13 Yield', 41), ('25 Road work', 36)]
```


```
100speed_blue.png :
[('1 Speed limit 30kmh', 21), ('1 Speed limit 30kmh', 16), ('7 speed limit 100kmh', 15)]

Computing  100speed_red.png :
100speed_red.png :
[('7 speed limit 100kmh', 44), ('5 Speed limit 80kmh', 35), ('2 Speed limit 50kmh', 31)]

Computing  100speed_yellow.png :
100speed_yellow.png :
[('40 Roundabout mandatory', 17), ('8 speed limit 120kmh', 17), ('8 speed limit 120kmh', 15)]

Computing  90speed_blue.png :
90speed_blue.png :
[('7 speed limit 100kmh', 74), ('7 speed limit 100kmh', 59), ('7 speed limit 100kmh', 48)]

Computing  90speed_red.png :
90speed_red.png :
[('7 speed limit 100kmh', 74), ('7 speed limit 100kmh', 59), ('7 speed limit 100kmh', 48)]

Computing  90speed_yellow.png :
90speed_yellow.png :
[('7 speed limit 100kmh', 74), ('7 speed limit 100kmh', 59), ('7 speed limit 100kmh', 48)]
```


```
Computing  nopassing_blue.png :
nopassing_blue.png :
[('9 No passing', 49), ('9 No passing', 46), ('9 No passing', 42)]

Computing  nopassing_red.png :
nopassing_red.png :
[('9 No passing', 96), ('9 No passing', 96), ('9 No passing', 80)]

Computing  nopassing_yellow.png :
nopassing_yellow.png :
[('9 No passing', 102), ('9 No passing', 98), ('9 No passing', 82)]
```

List of Work:

> Equal work was performed by both project members.

GitHub Page:

https://github.com/yu-ai-dev/COMP4102_Project

References

Han,Y., & Virupakshappa,K. (2015). *Hardware/Software Co-Design of a Traffic Sign Recognition System Using Zynq FPGAs.*Available from: https://www.researchgate.net/publication/286490452_HardwareSoftware_Co-Design_of_a_Traffic_Sign_Recognition_System_Using_Zynq_FPGAs

Dataset:

Joshi, S. (2019, October 30). *Road signs in Canada.* Retrieved April 17, 2021, from https://www.kaggle.com/stavanjoshi/road-signs-in-canada