Road Sign Preprocessing

Assignment 1

CS4053 Computer Vision

Alexandru Sulea
D Stream
#12315152
30 Octomber 2016

Contents

List of Tables

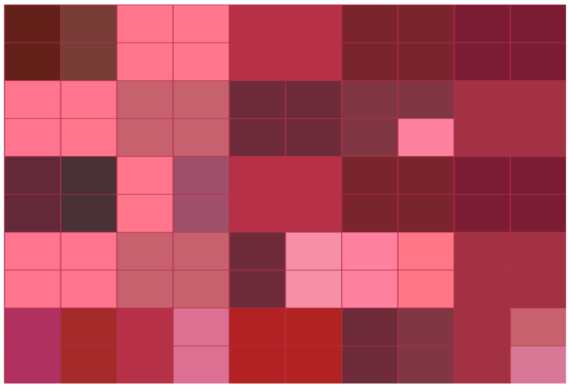
1 RED PIXEL DETECTION

1.1 Intro

In this section I outline my procedure and results for detecting red pixels. For this part of the assignment the lecture notes, recommended reading and online openCV recources to plan an adequate procedure for completing the assignment.

1.2 Procedure

Taking notes of what was discussed and shown in class together with online forms and the opencv page, I chose to plan out my procedure based on a histogram and backprojection detection model. To use the histogram though, red samples were needed to, in essence teach the opencv program what red is, and more specifically what type of red was required to be recognised for the assignment.
A histogram could not be constructed out out of the unaltered sample picture provided due to the program not understanding what it was required to look for.


(a) The second last red sample

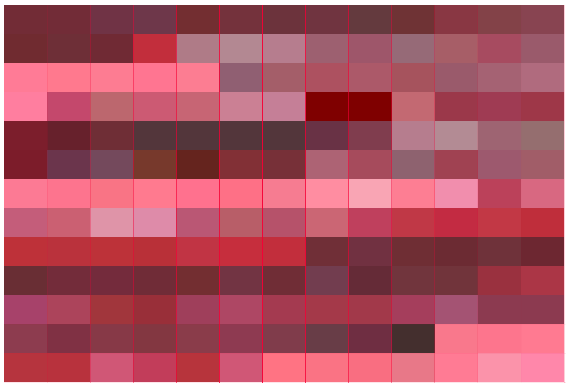

(b) The red sample image used for the current results

Figure 1: Figure of constructed red sample images

Thus, a new sample red image had to be created to make a histogram which would only contain the red present in road signs. To make the red sample image I used the photoshop program gimp to make a simple grid in which different red hues were stored. The red hue samples were taken from the sample road signs image. The images road signs were sampled at different points so as to get an even, accurate representation of the color red. The biggest problem with sampling the sample road signs image was the fact that a lot of the road signs had two, three or even more layers of shadowing on them. This created a problem which was only noticed when I started to paint in the various squares. I would, to my knowledge pick a red point in a sign only to discover that this point looked oddly brown or purple when transferred to my sample red image that was inserted into the histogram. These brown and purple points also caused a problem further down the road when dark bits of the background were identified as red due to those brownish squares.
Once the new sample red image had been created using different points on road signs in the sample road signs image the code could finally be written in.
The first step is to load in the images and check that the correct images have been loaded by the program without any exceptions or errors being thrown. That is where this part of the code comes in.

```
Mat roadsign_test2 = imread("filename.JPG", CV_LOAD_IMAGE_UNCHANGED);
```

```
if (roadsign_sample.empty()){
     printf("Cannot open video file: \n");
     return -1;}
  else { ...}
```

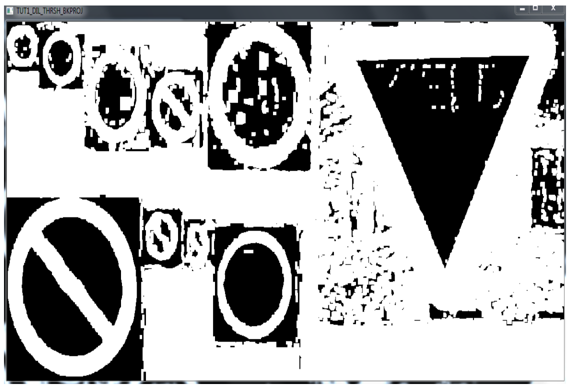
Next the image is converted to HSV or Hue Saturation Value so that we can separate the hue from the rest of the image. Since we are only interested in the color red and its multiple shades we will only be using hue, thus creating a 1D Red Hue Histogram.

```
//hue is from 0 to 180
     float huerange[] = {0, 180};
     //use only hue value
     hue_red.create(hsv_sample_red.size(), hsv_sample_red.depth());
     mixChannels(&hsv_test, 1, &hue_test, 1, ch, 1);
     /*Calculate and Normalize Hist*/
     calcHist(&hue_red, 1, ch, Mat(), hist_red, 1, &histSize_red, &ranges, true, false);
     normalize(hist_red, hist_red, 0, 255, NORM_MINMAX, -1, Mat());
     //drawing out the red histogram
     Mat histImg = Mat::zeros(w, h, CV_8UC3);
     for (int i = 0; i < bins; i++)
     {
        rectangle(histImg, Point(i*bin_w, h), Point((i + 1)*bin_w, h -
            cvRound(hist_red.at<float>(i)*h / 255.0)), Scalar(0, 0, 255), -1);
     }
     calcBackProject(&hue_test, 1, 0, hist_red, backproj, &ranges, 1, true);
     threshold(backproj, thresh_backproj, 8, 255, CV_THRESH_BINARY);
     dilate(thresh_backproj, dil_thresh_backproj, Mat(), Point(-1, -1), 2, 1, 1);
```

The above code shows how the red samples image was then loaded in, converted to HSV, then split up so that only the HSV would be used and finally used to make up the histogram.
Once the histogram had been set up all that remained was to back project the test image with the histogram. However the test image could only be back projected if it also had been converted to a hsv format and then had only its hue used. Once the two images were successfully back projected together a third image was created. This third image represented where the opencv program found red, thus the back projection. Red being the sample reds which were given to it.
For this project I conducted multiple tests with different and increasingly detailed red sample images due to the inaccuracy in my earliest back projected images. I found that the more red samples were given to the histogram the more accurate the histogram would become. This process has its limitations however, for example there could be red samples present in the test image that were not present in the sample image or vice versa. Thus 100 percent accuracy can never truly be achieved.
Once the image was back projected, it needed to be thresholded to remove any patches the program believed had a low probability of being red, such as orange brick. I then chose to dilate the pixels on the image so as to fill in empty spots in images and try to make them more accurate. I used dilate in an effort to also close the traffic sign circles so as they are later recognised as circles by floodfill. This process however had minimal success.
Morphology was also used in an effort to try and close the back projected round road signs. Morphology was better at closing holes and gaps, but for a best result process, I chose to use it in conjunction with dilate.

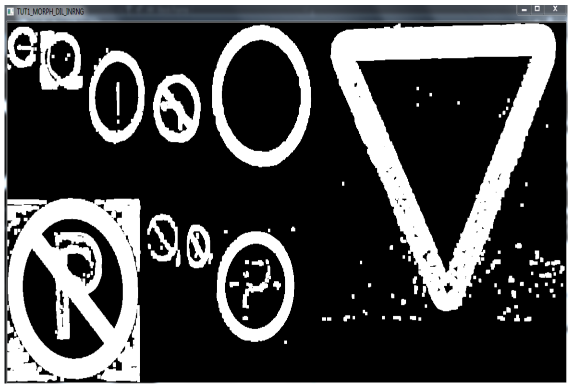
Despite my best efforts, I could not close the back projected circles due to the back projected image not recognising the red hue in some road signs. This did not affect the red samples but it did have a huge impact on the black and white pixel detection as floodfill only works on closed shapes, never opened ones, thus in effect any shape that was not closed would be lost in the process along with my accuracy.

To preserve the traffic sign shapes in my image throughout the contour hierarchy process I had to alter my final red pixel detection image to allow for the black and white pixel detection later on.



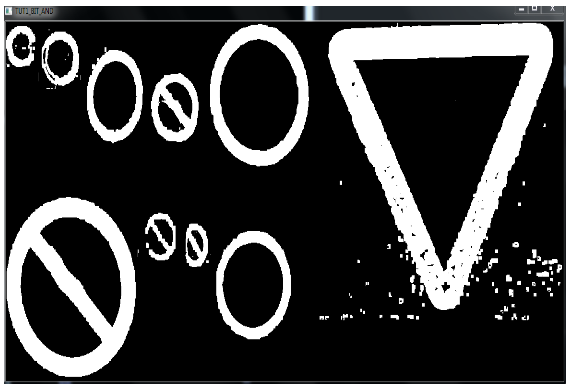
(a) The back projected image

+



(b) The inRange thresholded image

Figure 2: Figure of constructed tresholded images

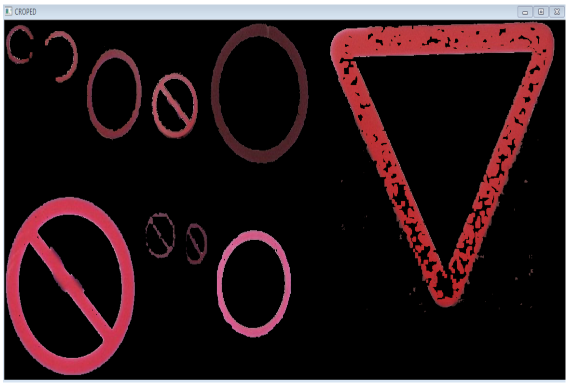


(a) The two images above ANDed give this result

Figure 3: Figure of constructed of the two tresholded images

```
inRange(hls_test, Scalar(10,10,55), Scalar(210, 190, 200), inrange_test);
dilate(inrange_test, dil_inrange_test, Mat(), Point(-1, -1), 2, 1, 1);
// Two images, back proj and inrange merged together
bitwise_and(dil_thresh_backproj, morph_dil_inrange_test, band_rng_bkprj);
dilate(band_rng_bkprj, band_rng_bkprj, Mat(), Point(-1, -1), 1, 1, 1);
/*Dilate and morphology again to close the remaining gaps*/
morphologyEx(band_rng_bkprj, band_rng_bkprj, MORPH_TOPHAT, element, Point(-1, -1), 50);
bitwise_and(roadsign_test,roadsign_test,redcrop, band_rng_bkprj);
```

The code above shows how I used inRange to get a similar but different thresholded image which has fuller contour of the road sign pictures but also a lot of other shapes in it. The thresholded image was also somewhat inaccurate, but by merging the two pictures together using AND a much clearer and more accurate representation of the red pixels in the image would be achieved. The image was then dilated and closed to help with remaining holes and breaks in the shapes. Finally the resulting image was used as a mask to construct a composite image to show how much of the red pixels from the original image were captured.


(a) A previous, less succesfull composite image using the previous (a) red sample image

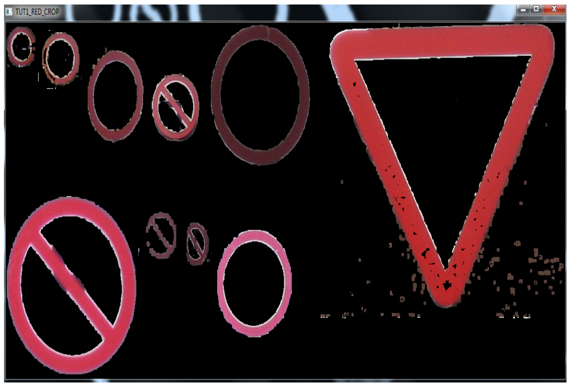

(b) The composite image showing the current red mixels detected

Figure 4: Figure of constructed tresholded images

```
CompareRecognitionResults(band_rng_bkprj, ground_red);
```

The above code shows the use of the function CompareRecognitionResults to compare the example thresholded red pixel image with the thresholded ground pixel image to achieve a score for the performance of my method.
The ground image is converted to HLS and then used inRange on it as discussed before. The only difference to this is that the ground image has only one overall value for each color present, making it very easy to threshold out the colors. Thus the following code was used to get the ground thresholded images of the three colors.

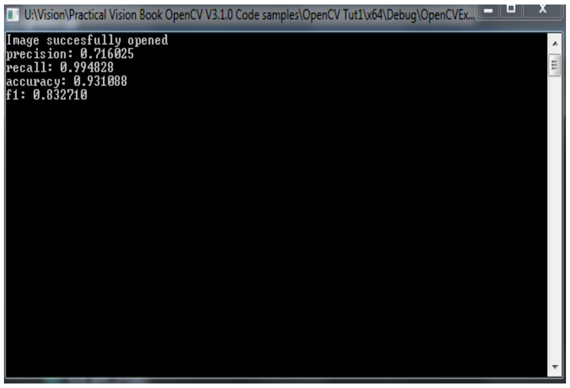
```
    inRange(hls_ground, Scalar(0, 0, 255), Scalar(0, 0, 255), ground_white);
    inRange(hls_ground, Scalar(0, 255, 255), Scalar(0, 255, 255), ground_red);
    inRange(hls_ground, Scalar(0, 0, 0), Scalar(0, 0, 0), ground_black);
```

1.3 Performance

The performance of the thresholded image was tested using the function provided in the practice tutorial. As seen in the picture and table, the precision for the color red is lower than that of black or white pixels. This is due in part to the difficulty in constructing a red sample image for the histogram. The innacuracy is also due to the dilation used on the shapes to connect the circles. The methadology helped in improving black and white pixel detection but at the cost of decreasing red pixel detection precison.

Table 1: RED PIXEL DETECTION VALUES

TYPE	VALUE
Precision	0.716025
Recall	0.994828
Accuracy	0.931088


U:\Vision\Practical Vision Book OpenCV V3.1.0 Code samples\OpenCV Tut1\x64\Debug\OpenCVEx...

```
Image succesfully opened
precision: 0.716025
recall: 0.994828
accuracy: 0.931088
f1: 0.832710
```

(a) The image shows the scoring achieved for the red pixels

Figure 5: The figure shows the scores for red pixel recognition

1.4 Discussion

One of the earliest problems noticed with hand sampling the red from the road signs was that the quality of the pictures was quite low. The pictures were also taken at an angle and with different cameras, thus the sharpness of the image also varied from picture to picture.

The performance for red pixels detection is high although it could have been higher. One major obstacle in a more precise detection algorythm is the quality of the pictures themselfes. Another being the background of the picture. While performing this assignment I have found it quite diffucult to separate a red road sign from its red brick background. Should even more red objects had been present in the picture, a more advanced form of Canny would have to be used to separate the objects and only then perhaps perform back projection.

By observing the threshold images provided for red pixel detection it can be clearly noticed that the picture has false positives, where it recognized parts of the background building as parts of the road sign. It also contains false negatives where, for some reason it did not recognize parts of the red borders in the road signs. These errors, which were due to the factors previously discussed, posed the biggest challenge to making an accurate road sign detecting opencv program.

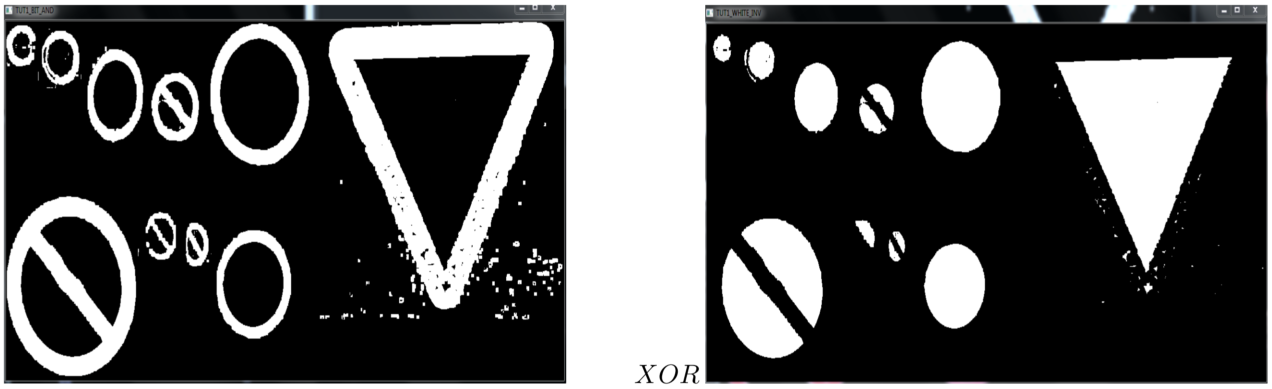
2 BLACK AND WHITE PIXEL DETECTION

2.1 Intro

In this section I outline my procedure and results for detecting black and white pixels.

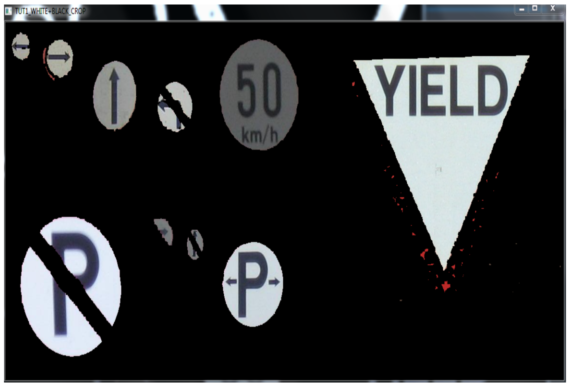
2.2 Procedure

To get the thresholded image of the white and black pixels a contour hierarchy process needs to be used. The white and black pixels cannot be back projected unfortunately due to the pictures being copy pasted on a white background, thus giving false positives for white pixels. The black pixels can not also be detected using black pixels due to the shadows and dark shapes present in the images, thus again, introducing false positives. Due to the extra work being done for red pixels, the thresholded red pixel image has closed shapes. Thus by using floodfilll, the inner space can be cropped away. Floodfill works by filling the inside of only closed border shapes in the red pixel thresholded image and then XORing the original red pixel thresholded image with the new flood filled image. The combination of these two images creates a composite which only has the inner space of the borders of the shapes in the thresholded red pixel image.
The problem now is that this image will encompass the white and black pixels inside the red border of the traffic sign.



(a) The composite thresholded image (b) The floodfilled image

Figure 6: The figure shows how floodfill works

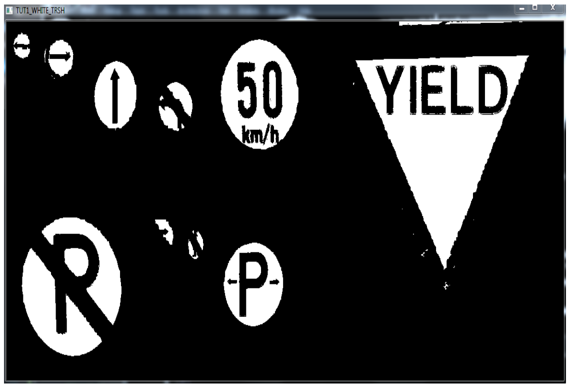


(a) The two images above ANDed and thenm used as a mask on the original color image give this result

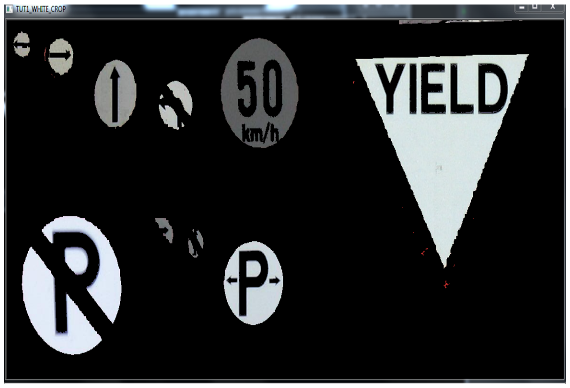
Figure 7: The figure shows how the contour hierarchy works

```
bitwise_not(white_ff, white_ff_not);
Mat white_out = (band_rng_bkprj | white_ff_not);
bitwise_xor(band_rng_bkprj, white_out, whiten);
bitwise_and(roadsign_test, roadsign_test, whiteblackcrop, whiten);
cvtColor(whiteblackcrop, wbcgrey,CV_BGR2GRAY);
threshold(wbcgrey, trsh_wht, 85, 255, CV_THRESH_BINARY);
bitwise_and(roadsign_test, roadsign_test, cropp_white, trsh_wht);
```

To separate black from white the new image needs to be ANDed with the original color image to get the location of the white and black pixel placement. Once a composite is created it is then grayscaled and thresholded, this will result in a new image which will only have the white pixels present.



XOR



(a) The composite thresholded image to get only the white outline

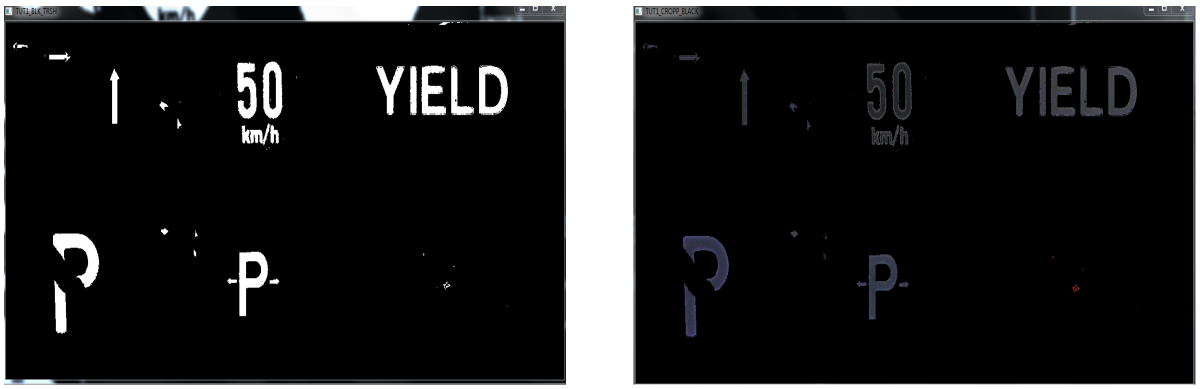
(b) The image in (a) used as a mask on the original color image to show how much of the wite pixels can be detected

Figure 8: The figure shows how contour hierarchy can be used to section off the white pixels.

The inner thresholded image is grayscaled to help with the thresholding but also to convert it to the appropriate CV_8UC1 format. If the image is not grayscaled the image matrix will be a different size than those of the ground image and thus will not be possible to be accurately compared, or as I found early on, to be even successfully inputted into the function.

```
Mat blk_ff = cropp_white.clone();
floodFill(blk_ff, cv::Point(0, 0), Scalar(0,0,0));
bitwise_not(blk_ff, blk_ff_not);
bitwise_and(whiteblackcrop, blk_ff_not, blk_trsh );
/*have to grayscale otherwise image format is not accepted*/
cvtColor(blk_trsh, blk_trsh, CV_BGR2GRAY);
threshold(blk_trsh, blk_trsh, 0, 255, CV_THRESH_BINARY);
```

To get a thresholded image of the black pixels the same process used to get white pixels is used again. Thus the white pixel thresholded image is XORed with the original black and white thresholded image, or a floodfilled white pixel thresholded image, they are the same thing, resulting in a composite image which has only the black road sign pixels sectioned off. This is then ANDed with the original color image, grayscaled so it has the appropriate format and thresholded again so that it can be inputted into the precision function.



(a) The composite thresholded image to get only the black pixels inside the road signs (b) The image in (a) used as a mask on the original color image to show how much of the black pixels can be detected

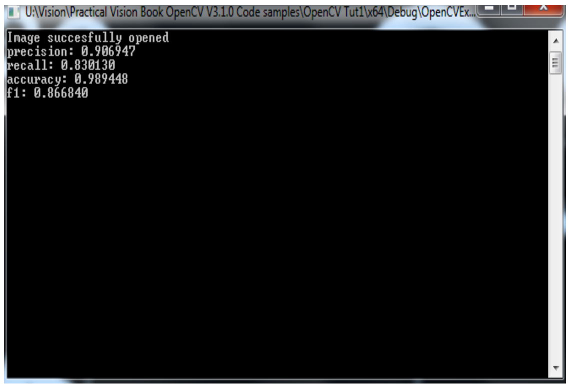
Figure 9: The figure shows how contour hierarchy can be used to section off the black pixels inside the road signs.

```
CompareRecognitionResults(blk_trsh, ground_black);
CompareRecognitionResults(scd_wht_cropp, ground_white);
```

The above code shows the last parts of the program in which the example thresholded images for white and black are compared with the thresholded images of the the ground image.

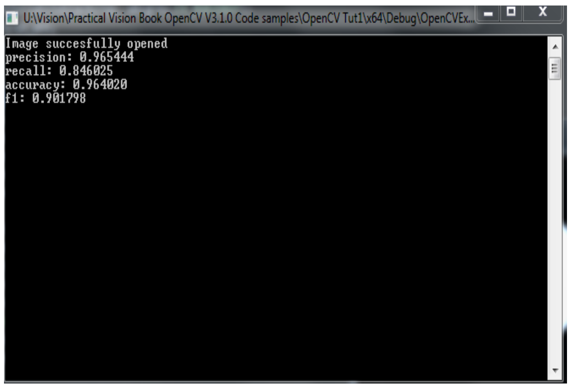
2.3 Performance

The performance of the white and black pixels was deduced from inserting the thresholded white and black pixels images into the Recognition function previously specified. As can be seen from the tables and images provided the precision on both types of pixels was significantly higher tan that of the red pixels.
The white pixels have the highest precision, this can be due to the fact that white is the easiest color to threshold out. The lower precision in blackpixels can be due to the fact that the thresholding process limitations may have been off, thus loosing some detail present in the process.



```
Image succesfully opened
precision: 0.906947
recall: 0.830130
accuracy: 0.989448
f1: 0.866840
```

(a) The image shows the scoring ac hieved for the black pixels



```
Image succesfully opened
precision: 0.965444
recall: 0.846025
accuracy: 0.964020
f1: 0.901798
```

(b) The image shows the scoring ac hieved for the white pixels

Figure 10: The figure shows the scores for the white and black pixel recognition

Table 2: The figure shows the scores for white and black pixel recognition

TYPE	WHITE VALUE
Precision	0.965444
Recall	0.846025
Accuracy	0.964020

TYPE	BLACK VALUE
Precision	0.906947
Recall	0.830130
Accuracy	0.989448

2.4 Discussion

The accuracy of the black and white pixels was somewhat a bit more accurate as there were far less similar background colors. Thus there was less interference and the background could be sectioned off more easily. The hue range to which the black and white pixels extended was also far narrower than that of the red pixels. By taking a look at the thresholded images provided for black and white pixels it can be seen that many of them contain false negatives. The lower part of a sign had been lost in the tresholding process due to its borders not being fully closed during flood fill.
Lastly, the biggest issue I encountered during this assignment was the values by which to threshold, dilate and erode. Unlike other functions in opencv, these processes require a human opinion. During the thresholding processes I struggled to understand why one lower or higher limit is better than another. My only conclusion from this process is that at the end of the day opencv requires a human to truly define what is useful and what is not. The values therfore that I used for my thresholding process were my personal input for what would make an accurate program.