Universidade do Porto

Faculdade de Engenharia

FEUP

Traffic sign detection

Project 1 report

Computer Vision

5th year of the Integrated Masters in Informatics and Computing Engineering

Author:

Carlos Miguel Correia da Costa – 200903044 – ei09097@fe.up.pt / carlos.costa@fe.up.pt

October 28, 2013

Index

1. Project specification

1.1. *Context*

Traffic sign recognition has several useful applications, ranging from simple driver assistance systems to full fledge autonomous vehicles.

They can help reduce accidents by alerting drivers of the presence of traffic signs that they may have missed due to bad atmospheric conditions or by their bad positioning in the road. Also, they are fundamental to make sure that autonomous vehicles can drive safely, and in coordination with other drivers.

1.2. *Project description*

This project will focus on the detection and recognition of speed limit signs, viewed at the street level and from a frontal perspective.

The traffic signs can be loaded from a stored image or video, and can also be from a live feed retrieved from a connected camera.

In order to improve the detection of the traffic signs as well as the recognition of the speed limit number, the images will be preprocessed to remove noise, improve contrast and equalize the color distribution.

1.3. *Project objectives*

This project aims to apply computer vision technics in a project with real applications, and test their effectiveness and their usefulness in image processing and analysis.

1.4. *Expected results*

After the detection is complete, the traffic signs will be segmented from the image, removing the background, and the recognized speed limit number will be highlighted on top of the respective traffic sign.

2. Project implementation

In the next sections it will be presented the main algorithms used, highlighting the reason for their use, and showing some examples on how they improved the detection and recognition of the speed limit signs.

In order to speed up the project development, the OpenCV library was used, since it provides an open source implementation of the most important algorithms that can be used in image processing and analysis.

The most important algorithms parameters can be adjusted at runtime using trackbars from the OpenCV HighGUI in order to allow the fine tuning of traffic signs with different lighting conditions and colors values.

2.1. *Image source*

The source of the images to be analyzed can be selected in the CLI, and can be provided as a path to an image or video, or by specifying the id of a connected camera.

2.2. Image preprocessing

2.2.1. Bilateral Filtering

To remove noise it is applied a bilateral filter, since it tries to smooth the image without losing its edges.

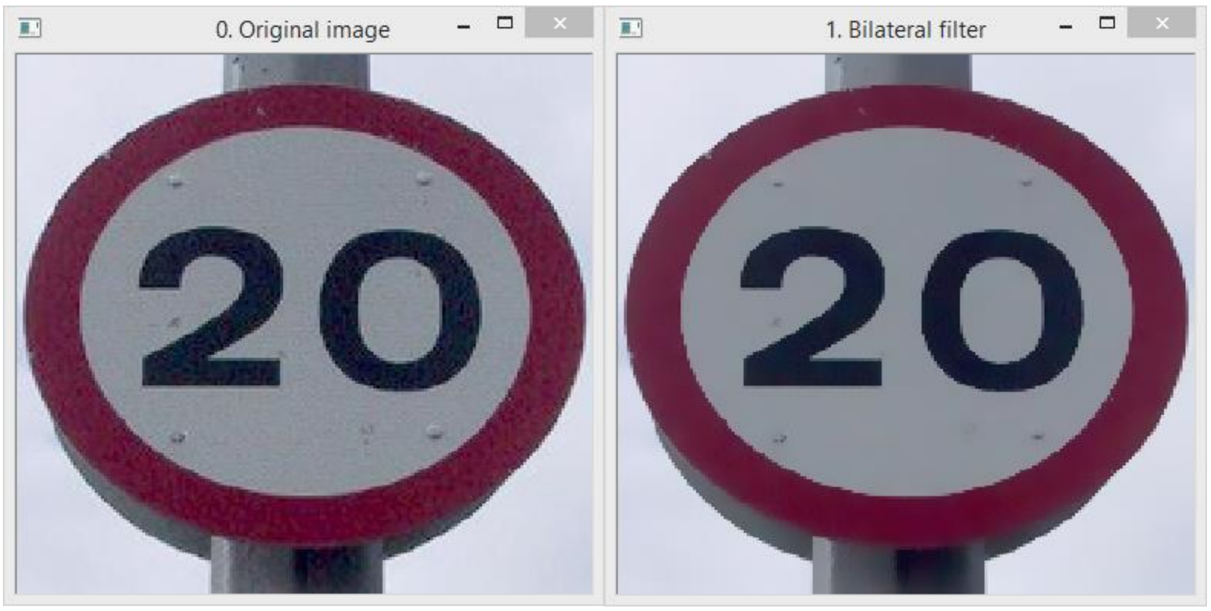


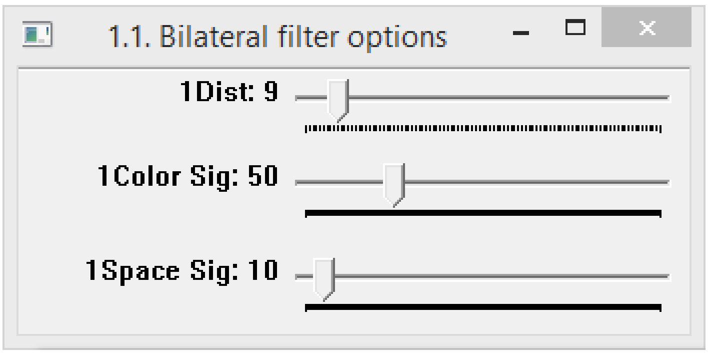
Fig. 1.1 – Effect of the bilateral filter in removing noise

✓ ***Parameterization***

a) **1Dist**: Diameter of each pixel neighborhood

b) **1Color Sig**: Filter sigma in the color space (larger value means farther color within the pixel neighborhood will be mixed together)

c) **1Space Sig*:*** Filter sigma in the coordinate space (larger value means that farther pixels will influence each other as long as their colors are close enough)



1Dist: 9

1Color Sig: 50

1Space Sig: 10

Fig. 1.2 – Bilateral filter parameters

2.2.2. Histogram equalization

To improve contrast and correct color distribution, it is used histogram equalization with the CLAHE method (contrast limited adaptive histogram equalization), in order to avoid the increase of noise in relation to the normal equalization (since it divides the space in a grid and applies equalization to each tile).

This is accomplished by converting the image to the YCrCb color space and equalizing the Y channel.

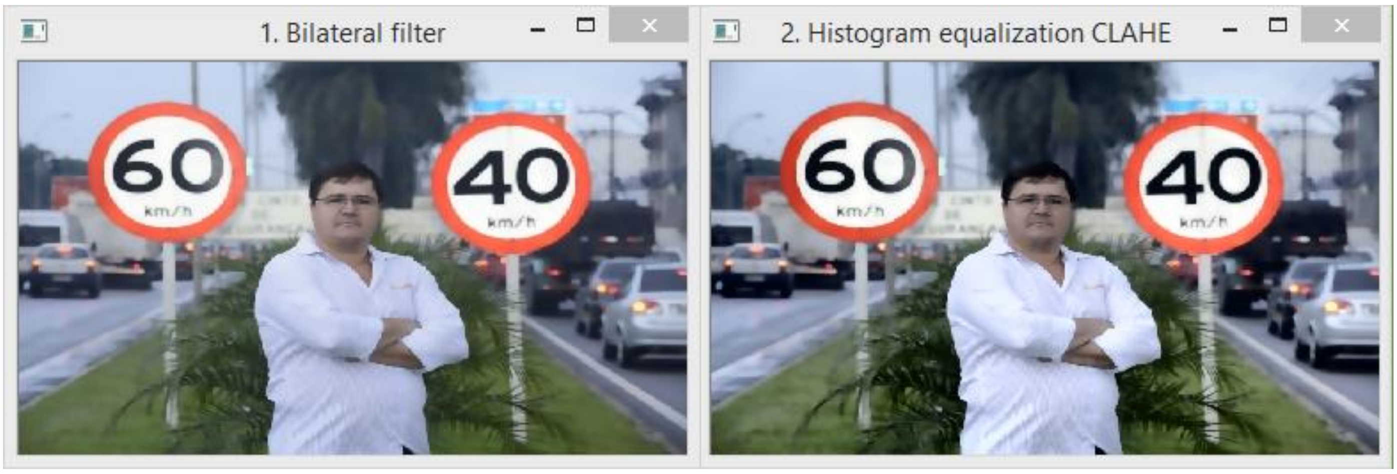
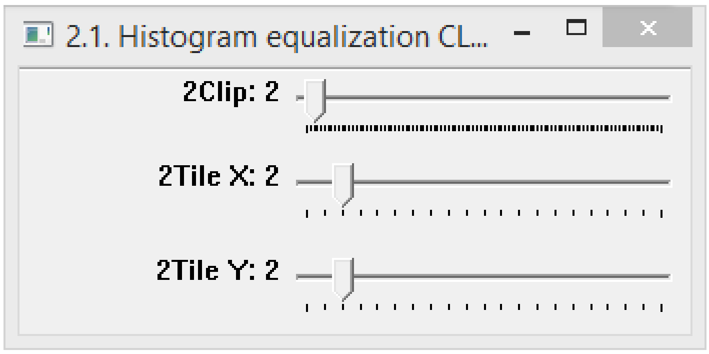


Fig. 2.1 – Impact of histogram equalization using CLAHE

✓ ***Parameterization***

a) **2Clip:** Threshold for contrast limiting

b) **2Tile X and 2Tile Y:** Size grid for histogram equalization



2Clip: 2

2Tile X: 2

2Tile Y: 2

Fig. 2.2 – CLAHE parameters

2.2.3. Contrast and brightness

Sometimes the histogram equalization can be improved or corrected by adjusting the contrast and brightness. This could be used to remove low contrast signals backgrounds.

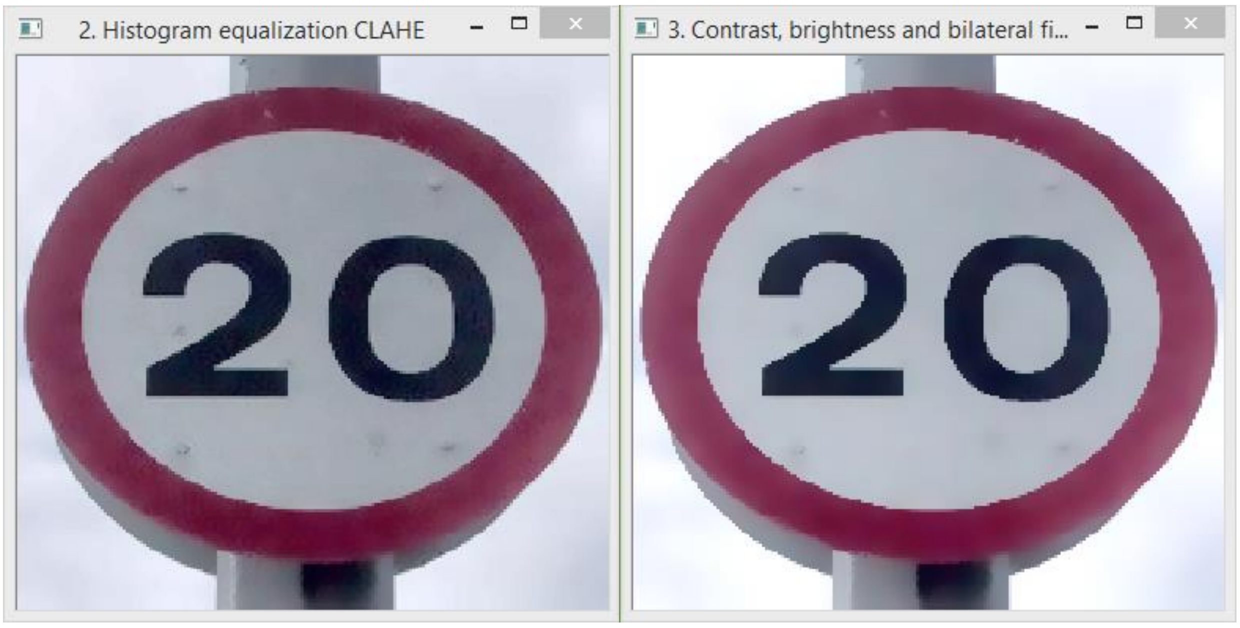
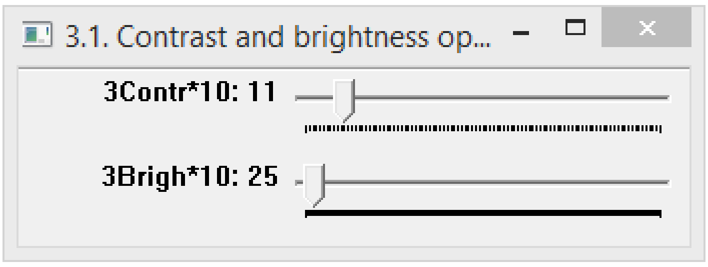

2. Histogram equalization CLAHE
3. Contrast, brightness and bilateral fi...

Fig. 3.1 - Removal of low contrast background and some noise by adjusting contrast and brightness (and applying bilateral filtering again)

✓ ***Parameterization***

a) **3Contr*10:** Contrast (the value used will be divided by 10)

b) **3Bright*10:** Brightness (the value used will be divided by 10)



3.1. Contrast and brightness op...

3Contr*10: 11

3Brigh*10: 25

Fig. 3.2 – Contrast and brightness parameters

2.3. *Color segmentation*

To detect the possible positions of speed limit signs, it is used color segmentation in the HSV color space, in order to create a binary image that detects the red regions of the speed limit sign.

To remove possible noise (small blobs) or aggregate blobs, it can be used morphological operations, such as opening and closing.

In the tests performed, using the opening operation proved very effective in removing small blobs of noise in images such as the one below, since the cars lights in the back are red, and they could appear in the color segmentation image.

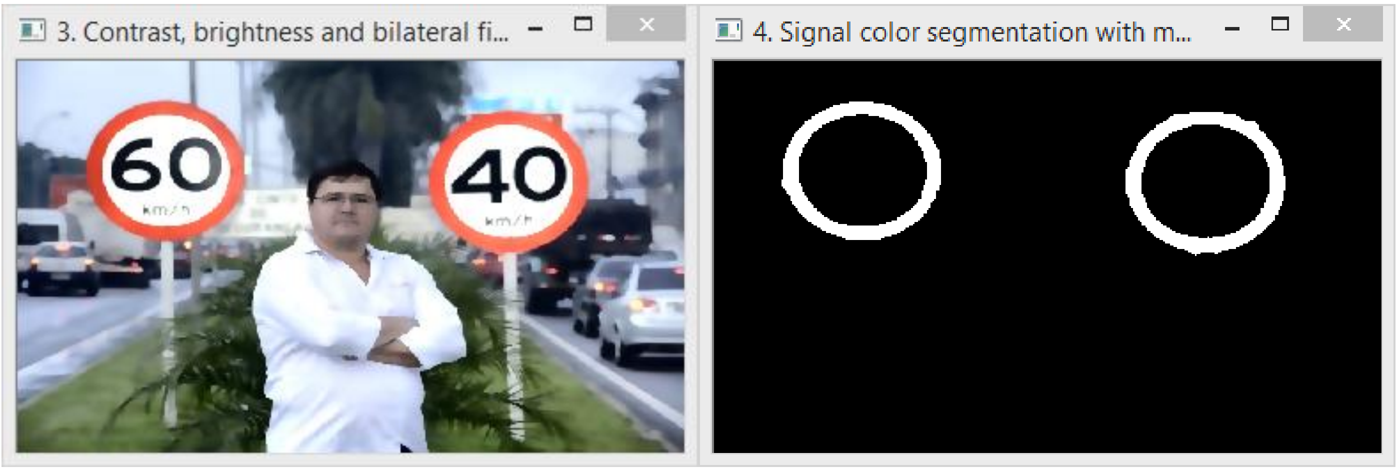

3. Contrast, brightness and bilateral fi...
60
km/h
40
km/h
4. Signal color segmentation with m...

Fig 4.1 – Color segmentation using HSV color space to identify speed limit circular red border

✓ ***Parameterization***

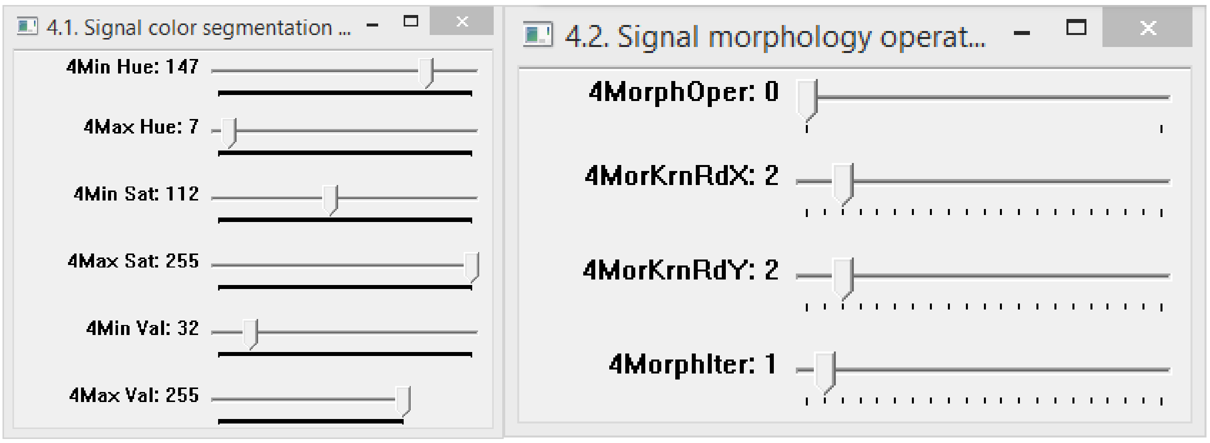
a) **Hue min and max:** from 0 to 180 in OpenCV corresponding to 0 to 360 in HSV (the hue value can wrap around, for example in the image below the hue goes from 147 to 180 and from 0 to 7)

b) **Saturation and value min and max:** from 0 to 255 in OpenCV, corresponding to 0 to 1 in HSV

c) **4MorphOper:** Morphological operation to use (0 to use opening, 1 to use closing)

d) **4MorKrnRdX and 4MorKrnRdX:** Structuring element in the format 2 * n + 1 (n is the value of the slider)

e) **4MorphIter:** Number of iterations to use in the morphological operation


4.1. Signal color segmentation ...
4.2. Signal morphology operat...

4Min Hue: 147
4Max Hue: 7
4Min Sat: 112
4Max Sat: 255
4Min Val: 32
4Max Val: 255

4MorphOper: 0
4MorKrnRdX: 2
4MorKrnRdY: 2
4MorphIter: 1

Fig. 4.2 – Color segmentation and morphology operations parameters

2.4. *Detection of sign by using HoughCircles and fitEllipse*

After having the possible positions of speed limit signals, it is used the Hough Circles transform in the color segmented binary image to select only the regions that have a circular shape.

Since the Hough Circles transform can give some false positives or return several circles for the same signal, after detecting the Hough Circles, it is performed a filtering phase, where the detected circles are aggregated in clusters and then each cluster is flatten to a single circle, computed from the mean of positions and radius of its elements (a circle belongs to a cluster if its center is inside one of the circles that belong to that cluster).

Besides the false positives problem, the Hough transform isn't very accurate in determining the center and the radius of each shape, and as such, to obtain a more precise segmentation, it is used the OpenCV fitEllipse function to an area around the Hough circle, in order to extract the speed limit signal from the background with high precision.

In the figure below we can see in blue the detected Hough Circle and in green the corresponding ellipse with its bounding rectangle (the detection of the Hough circles and ellipsis are performed in the color segmented binary image, and they are shown on top of a colored image only to evaluate the precision of the segmentation).

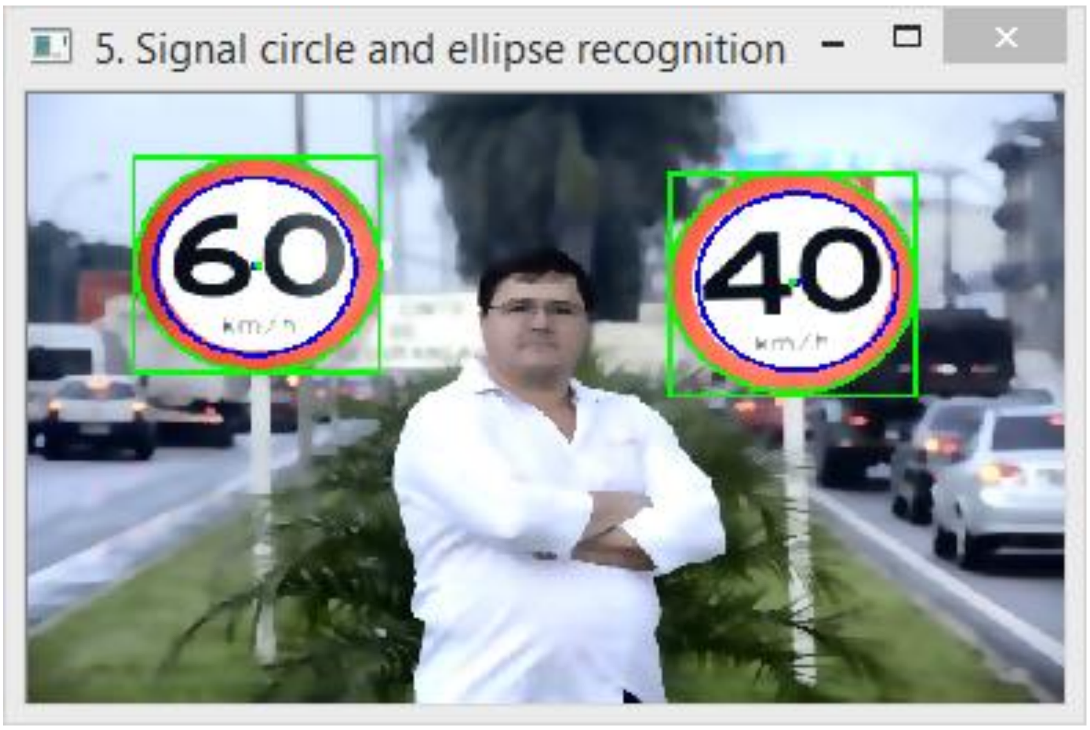


Fig. 5.1 – Segmentation of the signal from background using Hough Circles and fitEllipse

✓ ***Parameterization***

a) **5Hough DP:** Inverse ratio of the accumulator resolution to the image resolution

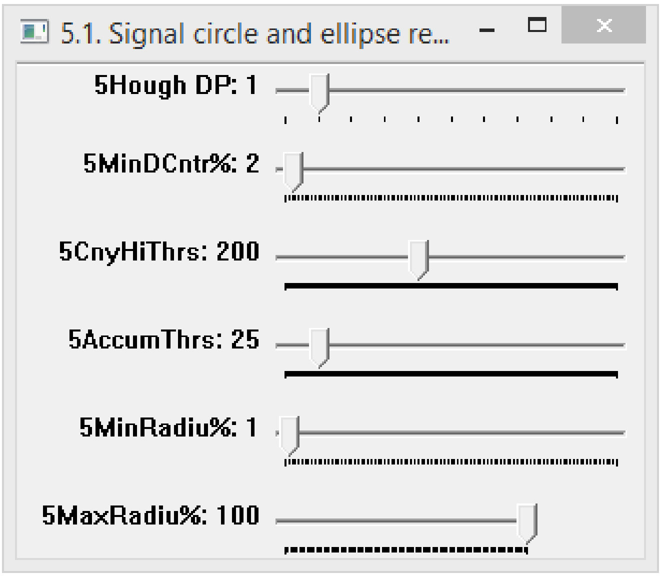
b) **5MinDCntr%:** Minimum distance between the centers of the detected circles

c) **5CnyHiThrs:** Higher threshold of the two passed to the Canny()

d) **5AccumThrs:** Accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected

e) **5MinRadius%:** Minimum circle radius percentage in relation to image

f) **5MaxRadiu%:** Maximum circle radius percentage in relation to image



5Hough DP: 1
5MinDCntr%: 2
5CnyHiThrs: 200
5AccumThrs: 25
5MinRadiu%: 1
5MaxRadiu%: 100

Fig. 5.2 – Hough Circles parameters

2.5. *Speed limit sign digits segmentation and recognition*

After having the regions of interest for each speed limit signal, the next phase is detecting the speed limit number.

To achieve this, the region inside each ellipse bounding box is extracted, the background outside the ellipse is removed, and then it is color segmented to extract the black regions associated with the digits.

After the color segmentation is complete, morphology operations are applied, such as opening and closing, to remove small blobs or merge large regions.

Next, with the text segmented in a binary image, each digit is separated with the OpenCV function findContours, and the 3 largest blobs (with enough percentage area and percentage height to be considered digits) are extracted.

To improve the recognition of the number, it was tested the skeletonization of the digits in the image (and digit templates). But since the digits in the image can be in different perspectives, it proved to be more effective to thin the digits (in the digit image and digit template) instead using the erode operator, in order to have more pixels to match in relation to the skeleton, but less pixels in relation to the original.

In the images below is presented an example of the digits color segmentation and digits thinning.

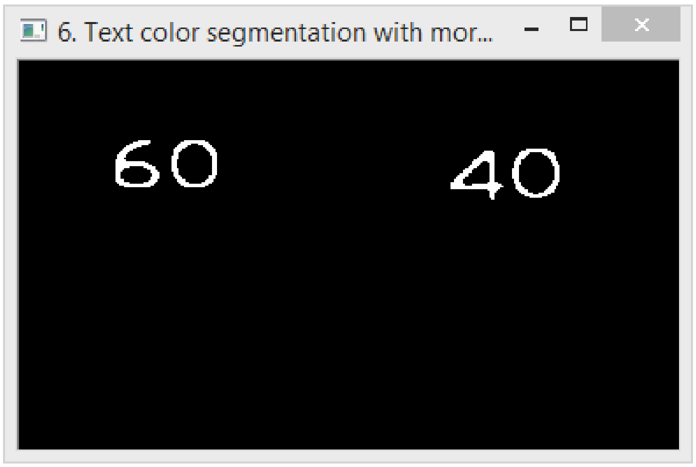


Fig. 6.1 – Digit color segmentation

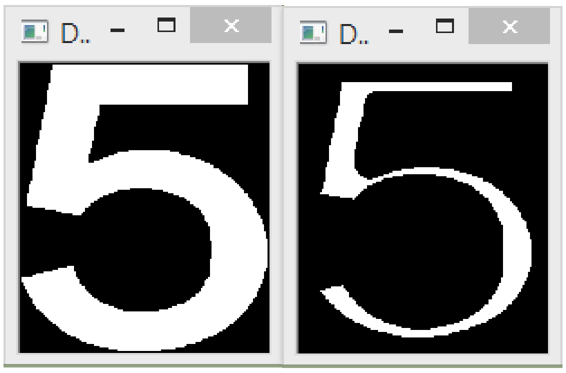


Fig. 6.2 – Digit before and after erosion operator

To improve the digit recognition, both the digit image and digit template are converted to the same resolution by resizing the larger image to match the size of the smaller image.

Having each digit properly segmented and preprocessed, the next phase is the recognition of each digit.

To achieve this, three different approaches were implemented.

The first aproach uses **template matching**, and has very good results when the digits are very well segmented, don't have gaps in the number lines and are in a frontal perspective.

The second aproach uses **feature detection** and has very good results even if part of the number is missing.

For this aproach several feature detectors were tested, namely SurfFeatureDetector, SiftFeatureDetector, FastFeatureDetector, GoodFeaturesToTrackDetector, OrbFeatureDetector, MserFeatureDetector and StarFeatureDetector. The one with the best results was GoodFeaturesToTrackDetector, and as such it is the one being used.

For the feature extration, it was tested the SurfDescriptorExtractor, SiftDescriptorExtractor, BriefDescriptorExtractor, FREAK and BRISK. The one with the best results was the SiftDescriptorExtractor, and as a result, it is the one being applied.

Finaly, for the matching descritor it was tested the FlannBasedMatcher and the BFMatcher. Since the FlannBasedMatcher obtained better results and was faster than the brute force matcher, it is the one that is being used.

After computing the feature matches, several algoritms were used to calculate the feature points that had good matches.

The first was based on an percentage offset from the minimum descriptor match distance. But it was including too many points that weren't good enough matches.

The second was implemented using knnMatch, to obtain two matches for each feature descriptor and then using the ratio between the match distances in each of these pairs to filter the good matches. But this method seemed to have the same problem that the previous one had.

As such, given that the speed limit signs aren't going to appear rotated, a third more restrictive method was used. This one uses a percentage threshold in relation to the pixel distance between the digit image and the template image. Meaming, if the position of the feature in the digit image is close to the position in the digit template, then it is probably a good match. This proved very effective since both the digit image and the digit template have the same resolution in pixels and the digits are in a frontal perspective. With this method, and with the right parameterization for the digit segmentation, it was achieved a very good recognition of the digits present in the images.

The third aproach was delegating the digit recignition to the **Tesseract library**. But since the project was meant to be developed with OpenCV, it won't be described in this report (although it is implemented).

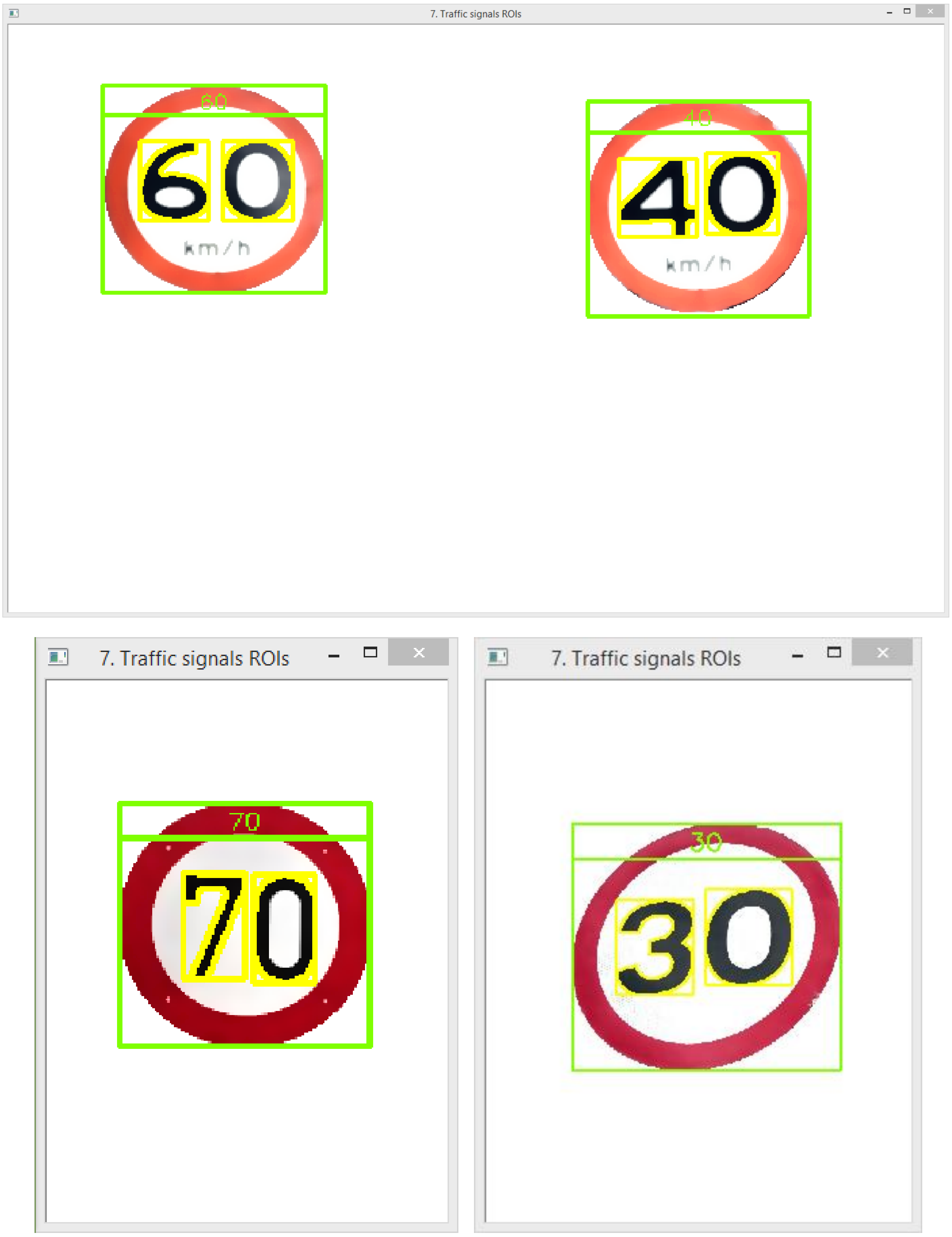


Fig. 6.3 - Result of the digit recognition obtained either with template matching or feature detection.

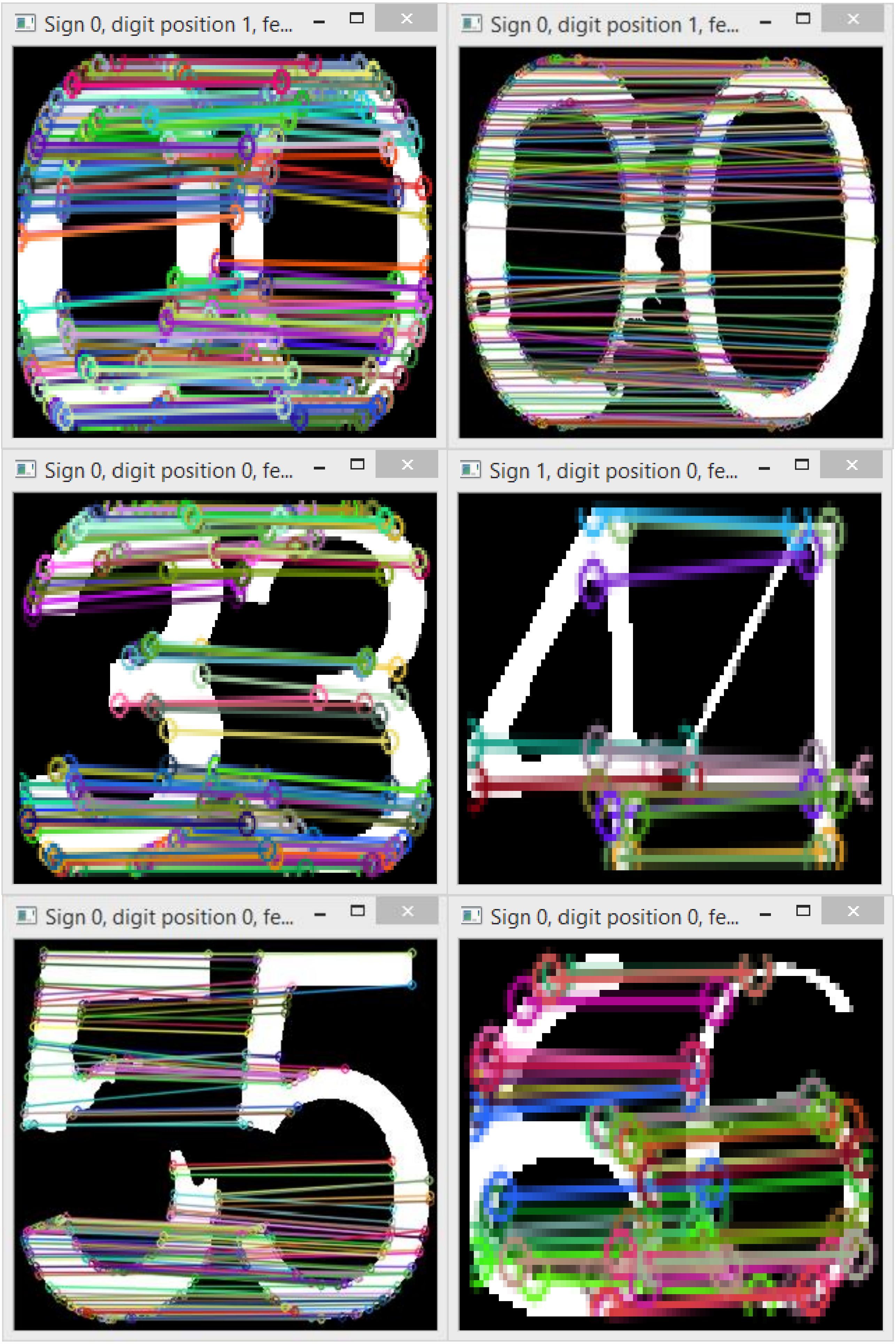

Sign 0, digit position 1, fe...
Sign 0, digit position 1, fe...
Sign 0, digit position 0, fe...
Sign 1, digit position 0, fe...
Sign 0, digit position 0, fe...
Sign 0, digit position 0, fe...

Fig. 6.4 – Examples of good matches obtained with the feature detection algorithm.

✓ ***Parameterization***

a) **Hue min and max:** from 0 to 180 in OpenCV corresponding to 0 to 360 in HSV (the hue value can wrap around)

b) **Saturation and value min and max:** from 0 to 255 in OpenCV, corresponding to 0 to 1 in HSV

f) **6MorphOper:** Morphological operation to use (0 to use opening, 1 to use closing)

c) **6MorKrnRdX and 4MorKrnRdX:** Structuring element for the morphological operation in the format 2 * n + 1 (n is the value of the slider)

d) **6MorphIter:** Number of iterations to use in the morphological operation

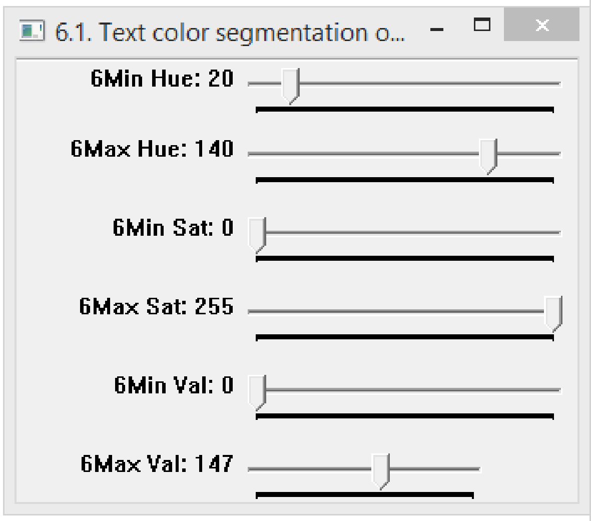
e) **6SklKrnRdX and 6SklKrnRdY:** Structuring element for the skeletonization / thinning operation for the digits in the format 2 * n + 1 (n is the value of the slider)

f) **6SkelIter:** Number of iterations to use in the morphological skeletonization / thinning operation

g) **6MinMatchP:** Minimum match probability to consider a digit correctly recognized (even low values can achieve good recognition, and can also be usefull to see what was the best digit matched when very low values are used)

h) **6DigRecMtd:** Digit recognition method to use (0 to use template matching, 1 to use feature detection and 2 to use the Tesseract library – isn't enabled by default. To enable is necessary to uncomment the USE_TESSERACT define in the ImageAnalysis.h file)

i) **6TMatchMtd:** Method to use when applying template matching (0 -> CV_TM_SQDIFF, 1 -> CV_TM_SQDIFF_NORMED, 2 -> CV_TM_CCORR, 3 -> CV_TM_CCORR_NORMED, 4 -> CV_TM_CCOEFF, 5 -> CV_TM_CCOEFF_NORMED)


6.1. Text color segmentation o...

6Min Hue: 20

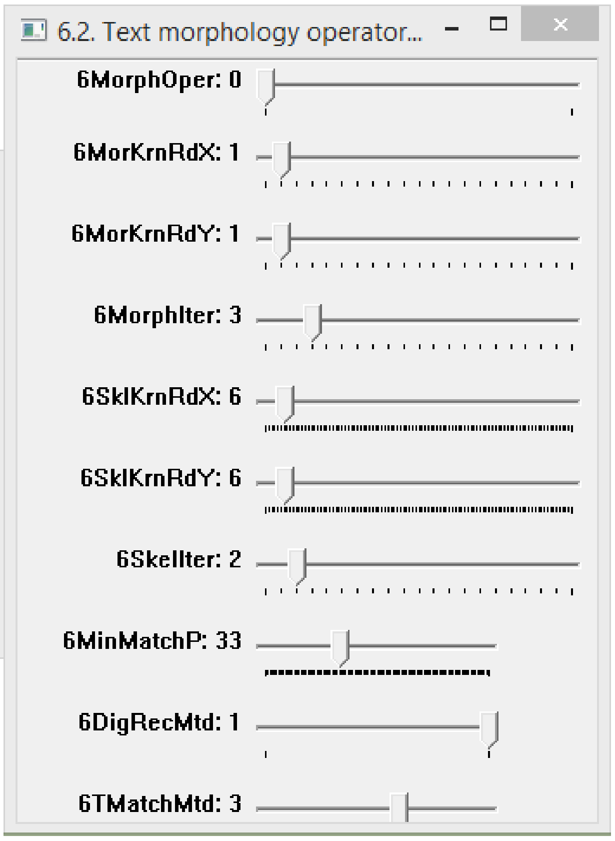
6Max Hue: 140

6Min Sat: 0

6Max Sat: 255

6Min Val: 0

6Max Val: 147


6.2. Text morphology operator...

6MorphOper: 0

6MorKrnRdX: 1

6MorKrnRdY: 1

6MorphIter: 3

6SklKrnRdX: 6

6SklKrnRdY: 6

6SkelIter: 2

6MinMatchP: 33

6DigRecMtd: 1

6TMatchMtd: 3

3. References

3.1. *Bibliography*

[1] Bradski, G., Kaebler, A., Lerning OpenCV, 2nd edition, O'Reilly, 2013

[2] Laganière, R., OpenCV 2 Computer Vision Application Programming Cookbook, Packt Publishing, 2011

[3] Baggio, D., Escrivá, D., Mahmood, N., Shilkrot, R., Emami, S., Levgen, K., Saragih, J., Mastering OpenCV with Practical Computer Vision Projects, Packt Publishing, 2012

[4] OpenCV Tutorials. Accessed on 28/10/2013

3.2. *Software used*

[5] OpenCV. Accessed on 28/10/2013

[6] Microsoft Visual Studio 2012. Accessed on 28/10/2013