Traffic sign detection and classification

Eduardo Correia
up20186433
João Martins
up201806436
Ricardo Fontão
up201806317

FEUP
University of Porto
Porto, Portugal

Abstract

Recognizing traffic signs accurately is becoming more and more necessary, with the development of driver assistance systems and automated driving. This process is usually composed of two main steps: detection of sign candidate regions and identification of signs.

The goal of this paper is to address a proposed solution, developed for the *Computer Vision* course, to the traffic sign detection and classification problem using classic computer vision methods.

This will include describing the chosen approach for each step of the developed system, such as image equalization, image segmentation, region of interest detection, and sign classification, which can identify and distinguish between: prohibition, obligation danger, information, stop and yield signs.

1 Introduction

The Python binding of the computer vision library OpenCV [2] was used for the development of the project. The Jupyter notebook containing the implementation can be found in Github.

Traffic sign detection and classification is a multi-layered problem that comprises several image processing and computer vision tasks. As such, the proposed solution tries to detect and identify signs in a process that is represented in the pipeline image 1.

The signs may present themselves in a variety of perspectives, with some being perpendicular to the optical axis of the camera while others are slanted. Besides perspective, illumination and visibility varies, posing a challenge in the segmentation phase of the proposed approach.

2 Traffic Sign Detection

The detection phase consists of three main phases: contrast enhancement and color normalization, color segmentation, and region of interest detection.

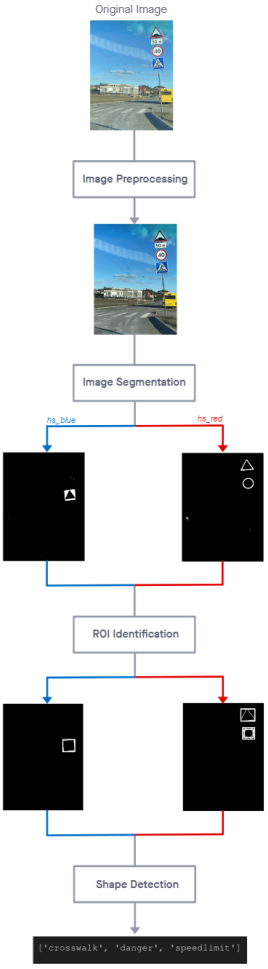


Figure 1: Pipeline

2.1 Image preprocessing

The designed approach aims to enhance regions of low visibility where signs may appear. The result of this stage compared with the original image can be seen in image 2

2.1.1 CLAHE

The first step is contrast enhancement, and the algorithm of choice was the CLAHE algorithm. Since at this point in the pipeline the working image is still colored, a conversion was needed for the histogram equalization to produce good results. The process involves converting the image from BGR to HSV, equalizing the saturation and value channels, and then converting back to BGR.

2.1.2 Mean-Shift

To aid in the segmentation step, the Mean-Shift clustering algorithm is used to make the images' colors more uniform and homogeneous. Multiple iterations of this algorithm were tested, which lead to better results but at a cost of a higher execution time. As a result, only one iteration is performed.

2.2 Color Segmentation

Color segmentation is a vital step of the developed solution. Its goal is to distinguish red and blue colored regions from the image that may contain traffic signs, in a noise-free and uniform manner. Since it produces filtered channels that will be the input of subsequent steps, badly segmented images will almost certainly lead to bad results.

The most intuitive color segmentation approach is to define a range of hue values in which pixels that don't fall into it are discarded. However, some images with lower saturation have signs that aren't included in the range, which will be wrongly ignored. To address this issue, two distinct color segmentation solutions were implemented and tested.

2.2.1 Red and Blue Scores

The first method, based on [5], assigns redness and blueness scores, depending on the distance of the color to the blue and red colors, to the image. This makes it so that a low value is assigned to areas close to ideal sign color range. This method uses the following formulas:

$$hd_{blue} = \begin{cases} 1 - \frac{|R-G|}{MAX-MIN}, & (MAX = B) \wedge (MAX - MIN \geq th) \\ 0, & otherwise \end{cases} \quad (1)$$

$$hd_{red} = \begin{cases} 1 - \frac{|G-B|}{MAX-MIN}, & (MAX = R) \wedge (MAX - MIN \geq th) \\ 0, & otherwise \end{cases} \quad (2)$$

(a) Before (b) After

Figure 2: Image preprocessing

$$sd = e^{\frac{(S-255)^2}{115^2}} \quad (3) \qquad hs = sd * max(hd_{blue}, hd_{red}) \quad (4)$$

The saturation measurement sd assigns a value to the pixels' saturation. It will be used to differentiate blue and red regions that have higher saturation values. With the red (R), green (G), blue (B) and saturation (S) channels, new channels are created: hd_{blue}, hd_{red} and hs. The hs channel, using hd_{blue} and hd_{red}, takes into consideration areas that have a high red or blue intensity, and, using the sd measurement, highlights pixels that have an increased saturation.

2.2.2 Shadow and Highlight Invariant Algorithm[3]

This method uses region growing to, on top of values that are within the desired hue range, include nearby shadowed or low saturated regions. The group employed a variation of this algorithm, as is presented in [3].

This is accomplished with the following steps:

1. Convert the BGR image to the HSV colorspace
2. Normalize H, S and V to $[0,255]$
3. Set hue values (H) that have a low saturation (< 40) to 0
4. Set hue values that (H) have high (> 230) or low brightness (< 30) to 0
5. Set all hue values (H) that are within the desired hue range to 255
   - The used hue range for blue signs is $]140,185]$
   - The used hue range for red signs is $[0,10[\cup]210,255]$
6. Divide the H image into 16×16 windows
7. For each window, if the number of white pixels (H is 255) is greater than 60, add the location of the first window's white pixel position to the seed point list.
8. For each seed point, apply a flooding algorithm to the H channel. Any seed point not yet processed that is contained within the filled area is removed from the seed point list.
9. Threshold the H image to only select areas that were filled by the flood fill algorithm

The thresholded image H is the output that can be used to find regions of interest and perform shape detection. This algorithm has the advantage of being somewhat resistant to shadows and saturation variation, since traffic signs that are in these areas are included due to the flooding step.

2.2.3 Result Comparison

A comparison between all three approaches can be found in image 3. As can be seen, the shadowed area in the prohibition sign is covered by the Shadow and Highlight Invariant, while not being represented using the Red and Blue Scores method.

For this reason, the Shadow and Highlight Invariant Algorithm was chosen as the segmentation algorithm. The blue and red channels are segmented into hs_{red} and hs_{blue}, which will be used in the subsequent steps.

2.3 Region of Interest Detection

The goal of this phase is to find areas of the image that have a high chance of containing traffic, i.e., regions of interest (ROI). This is a crucial step as signs that aren't covered in this phase will not go through the classification step, and thus, will be missed.

(a) Original (b) Red and Blue Scores (c) Shadow and Highlight Invariant Algorithm

Figure 3: Color segmentation

In addition, minimizing false positives in this step is also fundamental to achieving a good recall score.

Region of interest detection is performed both on hs_{red} and hs_{blue}, to identify the most relevant areas of each channel. The subsequent subsections will discuss the stages of ROI detection applied to each one.

2.3.1 Edge Detection

To increase the performance of contour drawing, edge detection is performed before it.

The initial approach used the Canny method with a *a priori* defined threshold followed by a morphological dilation in order to join edges that may have been wrongly separated. This, however, didn't produce the best results, as in some cases the dilation would join signs that were already well divided. Not using the dilation would imply misses on most crosswalk signs, due to the fact that the canny filter wasn't using the sign's white regions, which corresponded to less saturated areas, to draw edges.

Instead, an adaptive threshold method was used before applying the canny filter. This is calculated using the gaussian-weighted sum of the neighborhood values [1]. As expected, this solution also had much better results in images with varying illumination.

2.3.2 Contour drawing

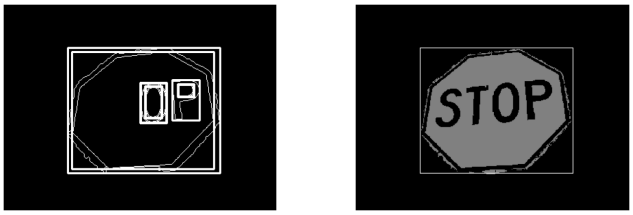
This is the main step of the ROI detection phase. It takes the image produced in the previous stage to draw contours around objects using the `drawContours` OpenCV function. It is important to mention that the `cv.RETR_TREE` flag was used to retrieve all drawn contours.

2.3.3 Bounding rectangle estimation and filtering

This step calculates, for each contour, a bounding rectangle around it. Afterward, to reduce false positives, contours that have a low probability of being signs are filtered out using a selection of predefined heuristics. These are:

- **Rectangle Minimum size** - All bounding rectangles that have a width or height less than 10 pixels are considered irrelevant, as this resolution isn't high enough for the classification stage.
- **Contour area higher than perimeter** - This filters out irregularly shaped contours.
- **Percentage contour to rectangle area** - Contours that occupy a small percentage of their bounding rectangle likely aren't signs
- **Rectangle aspect ratio** - The aspect ratio of the bounding rectangle must not be too unbalanced, as most traffic signs measured from a reasonable perspective are shaped like a balanced rectangle.
- **Rectangles contained inside rectangles** - Contours in which bounding rectangles are inside other bounding rectangles are filtered out, as traffic signs normally aren't contained inside other traffic signs.

The results of this ROI detection and filtering process can be seen in image 4.



(a) Before (b) After

Figure 4: ROI filtering

3 Traffic Sign Classification

After determining the regions of interest in the sign detection phase, the next step is to either classify the shape of the signal or discard the region, if no shapes are found.

The designed shape recognition process takes as input the contour and bounding rectangle of each ROI and uses them to predict one of four shapes: triangle, rectangle, octagon, or circle. At first, triangle and rectangle detection is performed, followed by circle and octagon detection if the previous fails. The ROI is classified as *unidentified* if both detection attempts fail.

3.1 Triangle and Rectangle detection

To classify danger, yield, and information signs there needs to be a distinction between upside down triangles, normal triangles, and rectangles.

For this purpose, a polygon is estimated, using `cv.approxPolyDP` function using each ROI's contours. Then, the number of lines in the polygon is used to determine whether the shape is a triangle or a rectangle.

To assess the orientation of a shape that is predicted as a triangle, two scores, one for a normal triangle and another for an upside down triangle, are calculated [5]. These scores are measured by dividing the ROI into regions according to the figure 5, and checking if there are any corners that fall within these regions. All corners are detected using the Harris corner detection method, using `cv.cornerHarris`.

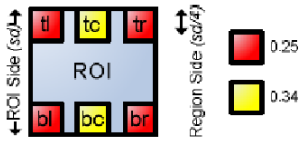


Figure 5: Corners ROI [5]

$$t_{up} = 1.32 \times (bl + br) + tc - 1.1 \times (tl + tr) \quad (5)$$

$$t_{down} = 1.32 \times (tl + tr) + bc - 1.1 \times (bl + br) \quad (6)$$

3.2 Circle and Octagon detection

Since obligation and prohibition signs are round, several circle shape detector algorithms were implemented and tested. This subsection will discuss briefly the developed approaches to solve this problem.

The initially considered approach used the Fast Radial Symmetry algorithm [4] as presented in [5]. This made it so that if a circle is present in the ROI, the output would be a high-intensity area in its center. The circle probability value, *cp*, can be measured by dividing the average of that intensity with the maximum obtained value. Since the OpenCV library doesn't implement this algorithm, the following implementation was adapted. However, even after parameter tweaking, results ended up being subpar: good quality obligation sign images would have a *cp* of 0.4. Another approach was needed.

The second approach tested based itself on the OpenCV's `minEnclosingCircle` method, which outputs the minimum enclosing circle of a specific contour. The circle probability of this method is calculated as the ratio between the area of the contour and the area of the circle. This, however, didn't perform well in images that had a titled sign, as the min enclosing circle estimation didn't cover the entire sign, leading to a misleading area ratio. In addition, given that the octagon shape is very similar to a circle, its circle probability is high, making stop signs and prohibition signs hard to distinguish using this method.

The third and final solution is based on a *circularity* measure, calculated by $\frac{4\times\pi\times area}{perimeter^2}$, which corresponds to a probability that a given shape is a circle, similar to *cp*. Instead of using a min enclosing circle, this method uses a convex hull for each ROI instead, using `cv.convexHull` and `cv.approxPolyDP`, The *circularity* can be tested to classify octagons (values between 0.7 and 0.96), and circles (values higher than 0.96), with values below 0.7, being considered undefined. This method produced satisfactory results, as it can distinguish prohibition signs from stop signs better than the previous approaches. However, in images where a speed limit sign is slightly tilted or partially hidden, the *circularity* resembles one of a stop sign, leading to incorrect predictions.

4 Results

To evaluate the designed pipeline, the following Kaggle dataset was used.

Its annotations only identify traffic lights, stop signs, speed limit signs, and crosswalk signs, even though its images include signs that belong to all types. This is a factor to take into account when analyzing the recall metric, as it may be inaccurate due to missing annotations.

The first type of images to analyze is the images where the sign is perpendicular to the optical axis of the camera and there are no obstructions or illumination artifacts. A specific example can be found at figure 6. For this specific image, the system predicted a stop sign correctly.

The second type of images analyzed were images with desaturated colors, making red look almost black in some cases. This makes the color segmentation process much harder. This type of image also includes dirty images where there are mild occlusions to the signs. A specific example can be found at figure 7. For this specific image, the system couldn't detect a single sign failing at the color segmentation stage.

The last type of image to analyze is the one where, for example, some prohibition signs have a certain blue tint on top of them, again disrupting the color segmentation process. A specific example can be found in figure 8, where the system predicted a stop sign instead of two prohibition signs.

For the total dataset, the results for the accuracy and recall for each type of sign can be found in table 1.

Sign	Precision	Recall
Stop	0.76	0.44
Crosswalk	0.93	
Speed limit	0.41	

Table 1: Performance metrics statistics

5 Conclusion

In conclusion, our recognition process gives a reasonable performance, correctly identifying most of the well-represented signs and also handling some edge cases.

Regarding future work, we could expand our detection to other signs not currently covered, such as traffic lights, and identify the information present in the detected signs, such as the speed limit number. In addition, since the designed solution struggles to distinguish between some stop and speed-limit signs, a better circle shape detection method could be employed. A possible alternative could use the stop letters to perform feature matching with a stop sign reference. The ROI detection phase could also be improved, as the number of false positives is high. Stricter filtering may solve this, but it may come at a cost of accuracy.

References

[1] Image thresholding. URL `https://docs.opencv.org/4.5.5/d7/d4d/tutorial_py_thresholding.html`.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Hasan Fleyeh. Traffic and road sign recognition. 2008.

[4] G. Loy and A. Zelinsky. Fast radial symmetry for detecting points of interest. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):959–973, 2003. doi: 10.1109/TPAMI.2003.1217601.

[5] Carlos Filipe Paulo and Paulo Lobato Correia. Automatic detection and classification of traffic signs. In *Eighth International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS '07)*, pages 11–11, 2007. doi: 10.1109/WIAMIS.2007.24.

6 Annex

Figure 6: Perfect Image

Figure 7: Desaturated Image

Figure 8: Color-shifted Image