Traffic Sign Detection in Road Scenes

Vonica Ioan-Paul

1 Project Purpose

The project aims to develop a software application that detects and marks traffic signs in static traffic images. The targeted sign types are:

- **STOP**: A red octagonal sign with "STOP" text, requiring complete vehicle stop.
- **YIELD**: An equilateral triangle with its point facing down, usually red and white, indicating the obligation to yield to other traffic participants.
- **NO ENTRY**: A red circle with a horizontal white band, prohibiting vehicle access in that direction.
- **SPEED**: A white circle with a red border and a number in the center, indicating the maximum permitted speed.

The solution uses classic image processing methods, such as color-based segmentation, contour detection, and geometric classification, to identify and label these signs. This type of system can be integrated into advanced driver assistance systems (ADAS) or used for inventory and verification of road signs in cadastral surveys.

2 Algorithmic Principles and Theoretical Foundation

2.1 HSV Color Space

The algorithm uses the HSV (Hue-Saturation-Value) color space instead of RGB for color segmentation, as HSV separates color information (hue) from intensity (value) and saturation. This offers the following advantages:

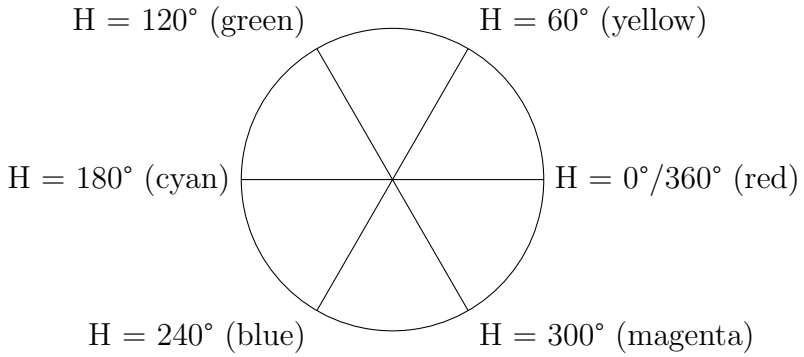

Figure 1: Circular representation of the H (Hue) component in HSV space

- **Resistance to illumination variations**: In HSV, the V (Value) component captures brightness information, allowing segmentation based only on H and S.
- **Intuitiveness**: Hue (H) is represented circularly (0-360°), where red appears both at the beginning (0-10°) and at the end (160-180°). This property facilitates efficient red color detection.
- **Precise segmentation**: By defining an interval for H and minimum thresholds for S and V, we can efficiently isolate red regions, even under variable lighting conditions.

In the HSV space implemented in OpenCV, the H component is scaled from 0° to 180° (not 360°), and the intervals for the red color are approximately 0-10 and 160-180.

2.2 Gaussian Filtering

The Gaussian filter is applied as a pre-processing step for:

- **Noise reduction**: Eliminates high-frequency intensity variations that can affect segmentation.
- **Blurring fine details**: Smooths textures and patterns that could fragment the segmentation.
- **Improving region coherence**: Contributes to forming more homogeneous regions following segmentation.

The Gaussian filter uses a two-dimensional normal distribution to calculate the kernel weights:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

where σ represents the standard deviation of the distribution, which in our implementation has a value of 2.

2.3 Morphological Operations

After red color segmentation, we apply two essential morphological operations:

1. **Closing operation (MORPH_CLOSE)**: It is a dilation followed by an erosion, which:
   - Removes small holes inside segmented regions
   - Connects fragmented regions that are close to each other
   - Smooths the contour without significantly modifying the region size
2. **Opening operation (MORPH_OPEN)**: It is an erosion followed by a dilation, which:
   - Removes isolated points and thin protrusions
   - Separates regions connected by a narrow bridge

- Reduces noise while preserving the shape and size of significant elements

We use elliptical structuring elements of different sizes (5×5 for closing and 3×3 for opening) to adapt the operations' effect to the image scale and the expected size of the signs.

2.4 Polygonal Approximation of Contours

The `approxPolyDP` algorithm simplifies complex contours into polygons with fewer vertices while preserving the essential shape. The method implements the Douglas-Peucker algorithm, which works as follows:

1. Connects the first and last points of the contour with a straight line
2. Identifies the point furthest from this line
3. If the point's distance exceeds a specified threshold (in our case, 2% of the contour perimeter), that point is kept
4. The process repeats recursively for the resulting line segments

This polygonal approximation is crucial for determining the number of vertices of the signs (3 for a triangle, 8 for an octagon) and for simplifying subsequent geometric analysis.

2.5 Geometric Features for Classification

The algorithm extracts and uses several geometric features to differentiate between sign types:

2.5.1 Circularity

Circularity measures how close a contour's shape is to a perfect circle and is calculated according to the formula:

$$\text{Circularity} = \frac{4\pi \cdot \text{Area}}{\text{Perimeter}^2} \tag{2}$$

A perfect circle has a circularity of 1.0, and any other shape has a lower value. In our implementation, we use a threshold of 0.80 to identify circular signs (SPEED and NO ENTRY).

2.5.2 Red Fill Ratio

This feature represents the proportion of red pixels inside the contour and is essential for differentiating between signs with similar shapes but different color distributions:

$$\text{RedFillRatio} = \frac{\text{Number of red pixels}}{\text{Total number of pixels in contour}} \tag{3}$$

Characteristic values for each sign type are:

- STOP: > 0.85 (predominantly red)

- NO ENTRY: moderate value (0.35-0.85)
- SPEED: < 0.35 (predominantly white with red border)

2.5.3 Centroid and Orientation

For the YIELD sign (triangle), we determine orientation by comparing the position of the top vertex with that of the centroid. A triangle with its point down will have the top vertex above the centroid.

The centroid (C_x, C_y) is calculated using geometric moments:

$$C_x = \frac{M_{10}}{M_{00}}, \quad C_y = \frac{M_{01}}{M_{00}} \tag{4}$$

where M_{pq} represents the moment of order $(p + q)$:

$$M_{pq} = \sum_{x,y \in \text{Region}} x^p y^q \tag{5}$$

3 Detailed Solution Description

3.1 Image Pre-processing

The pre-processing stage prepares the image for robust red color segmentation:

1. **Image loading**: The source image is read from a file and checked for validity.
2. **Applying the Gaussian filter**: A 5×5 Gaussian kernel with $\sigma = 2$ is applied to reduce noise and blur fine details.
3. **Conversion to HSV space**: The image is transformed from RGB to HSV to facilitate hue-based segmentation.

3.2 Red Color Segmentation

Segmentation identifies red regions that may contain traffic signs:

1. **Defining color ranges**: Two binary masks are created to capture the entire red spectrum:
   - First mask: H: 0-10, S: 60-255, V: 80-255 (orange-red)
   - Second mask: H: 160-180, S: 60-255, V: 80-255 (purplish-red)
2. **Combining masks**: The two masks are combined with a logical OR operation to obtain the complete red mask.
3. **Mask enhancement**: Morphological operations are applied to clean and consolidate regions:
   - Closing operation with a 5×5 elliptical structuring element fills small holes
   - Opening operation with a 3×3 elliptical structuring element removes noise and small artifacts

3.3 Contour Detection and Processing

This stage identifies potential sign shapes from the binary mask:

1. **Contour extraction**: The `findContours` function identifies all external contours from the binary mask, using the `CHAIN_APPROX_SIMPLE` method for efficient storage.
2. **Area-based filtering**: Contours with an area less than 500 pixels are eliminated to reduce false positives.
3. **Perimeter calculation**: For each valid contour, the perimeter is determined using `arcLength`.
4. **Polygonal approximation**: The contour is simplified into a polygonal shape using the Douglas-Peucker algorithm, with a threshold of 2% of the perimeter.
5. **Moments and centroid calculation**: Geometric moments are calculated to determine the center of mass of each detected shape.
6. **Feature extraction**: For each contour, the following is calculated:
   - Circularity: $\frac{4\pi \cdot \text{Area}}{\text{Perimeter}^2}$
   - Red fill ratio: proportion of red pixels inside the contour
   - Number of polygon vertices: `approx.size()`
   - Convexity: checked with `isContourConvex`

3.4 Sign Classification Algorithm

Classification uses a rule-based system based on the extracted features:

Classification rules are optimized to differentiate the four targeted sign types:

1. **YIELD**:
   - Number of vertices: exactly 3 (triangle)
   - Orientation: the topmost vertex is above the centroid (triangle pointing down)
2. **STOP**:
   - Number of vertices: exactly 8 (octagon)
   - Convexity: convex contour
   - Red fill ratio: > 0.85 (predominantly red)
3. **Circular signs** (circularity > 0.80):
   - **SPEED (Speed limit)**: red fill ratio < 0.35 (predominantly white)
   - **NO ENTRY (No entry)**: red fill ratio $>= 0.35$ (significant amount of red)

Algorithm 1 Traffic Sign Classification

for each contour c in the contour list **do**
    Calculate area $\leftarrow$ contourArea(c)
    if area < 500 **then**
        continue with the next contour
    end if
    Calculate perimeter $\leftarrow$ arcLength(c, true)
    Obtain approximate polygon *approx* using approxPolyDP
    Calculate centroid *cent* using moments
    Calculate circularity $\leftarrow 4\pi \cdot \text{area}/\text{peri}^2$
    Calculate redRatio $\leftarrow$ redFillRatio(*approx*, redMask, size)
    if *approx* has 3 vertices isTrianglePointingDown(*approx*) **then**
        Classify as "YIELD"
    else if *approx* has 8 vertices isContourConvex(*approx*) redRatio > 0.85 **then**
        Classify as "STOP"
    else if circularity > 0.80 **then**
        if redRatio < 0.35 **then**
            Classify as "SPEED"
        else
            Classify as "NO ENTRY"
        end if
    end if
    if classification successful **then**
        Draw the contour and add label centered on centroid
    end if
end for

3.5 The `isTrianglePointingDown` Function

This function determines a triangle's orientation by comparing the position of its top vertex with that of the centroid:

1. **Moments and centroid calculation**: The triangle's center of mass is determined.
2. **Top vertex identification**: The three points are scanned to find the point with the minimum y coordinate (highest on the image).
3. **Position comparison**: It checks if the top vertex has a y coordinate less than the centroid's y coordinate.
4. **Result interpretation**: If the top vertex is above the centroid, the triangle is pointing down (YIELD sign).

This check is essential to avoid confusion with other triangular signs that have different orientations.

3.6 The `redFillRatio` Function

This function determines the proportion of red pixels inside a contour, essential for differentiating between signs with similar shapes:

1. **Creating the polygon mask**: A binary mask is created in which the polygon is filled with the value 255 (white).
2. **Intersection with the red mask**: A logical AND operation is performed between the polygon mask and the red mask.
3. **Pixel counting**: The non-zero pixels are counted in both the intersection and the polygon mask.
4. **Ratio calculation**: The number of red pixels inside is divided by the total number of pixels in the polygon.

These red fill ratios allow precise differentiation between speed limit signs (predominantly white) and no entry signs (with a higher proportion of red).

3.7 Visualization of Results

In the final stage, the detected and classified signs are marked on the original image:

1. **Drawing contours**: The polygonal contours of the signs are drawn with specific colors (green for STOP and YIELD, red for SPEED, blue for NO ENTRY).
2. **Adding labels**: Text labels are added centered on the centroid of each detected sign.
3. **Displaying the result**: The final image with detected signs is displayed in a window.

4 Solution Analysis and Improvement Perspectives

4.1 Advantages of the Implemented Method

- **Computational efficiency**: Using color-based segmentation and simple geometric rules allows for rapid processing, potentially applicable in real-time systems.
- **Robustness to scale variations**: The geometric features used (circularity, fill ratio, number of vertices) are scale-invariant, allowing detection of signs of different sizes.
- **Conceptual simplicity**: The rule-based approach is easy to understand and modify, without requiring extensive training datasets.
- **Extensibility**: The system can be extended to include other signs by adding additional rules and features.

4.2 Limitations and Challenges

- **Sensitivity to lighting conditions**: Color-based segmentation can be affected by shadows, reflections, or extreme lighting conditions.
- **Color dependency**: The current implementation relies heavily on red color detection, limiting applicability to signs containing red.
- **Vulnerability to occlusion**: Partially covered signs may have modified contours, affecting feature extraction and classification.
- **Distorted geometry**: Oblique viewing angles can distort the perceived shape of signs, affecting the number of vertices and circularity.
- **Absence of text recognition**: The system cannot differentiate between various speed limit signs (30, 50, 70 km/h etc.), treating them all as "SPEED".

4.3 Improvement Directions

- **Multi-color detection implementation**: Extending segmentation to include other colors (blue, yellow, white) would allow detection of a wider range of signs.
- **OCR (Optical Character Recognition) integration**: Adding text recognition to identify specific speed limits and other textual information.
- **Perspective correction**: Implementing perspective transformations to normalize sign shapes viewed from oblique angles.
- **Using machine learning methods**: Integrating machine learning classifiers (SVM, CNN) could improve classification accuracy and robustness to variations.
- **Temporal filtering**: For video applications, implementing temporal filtering to reduce false detections and improve classification consistency between frames.
- **Distance estimation**: Adding distance estimation to detected signs, using information about the apparent size of signs.

5 Example Images for Detected Signs

This section presents visual examples for each sign type, used for testing and validating the algorithm.

Figure 2: STOP - red octagonal sign with white text

Figure 3: YIELD - red triangle with white border, point down

Figure 4: NO ENTRY - red circle with horizontal white band

Figure 5: SPEED - white circle with red border and number in center

6 Conclusions

The implemented system demonstrates the effectiveness of classical image processing methods in detecting and classifying traffic signs. By combining color-based segmentation, geometric analysis, and classification rules, the algorithm can successfully identify four common types of road signs: STOP, YIELD, NO ENTRY, and SPEED.

The solution offers a good balance between complexity and performance, being suitable for educational applications and simple driver assistance systems. With the suggested improvements, especially the integration of machine learning techniques and the addition of text recognition.