Final Project
ROAD SIGNS AND TRAFFIC LIGHTS DETECTION IN DRIVING SCENES

Giacomo Calabria
Computer Vision - University of Padua
22 January 2025

CONTENTS

***Abstract*—We have investigated various computer vision techniques to recognize traffic lights and road signs in driving scenes. Using the Open-CV library and YOLO-based models, we have analysed various pictures and we have compared different approaches.**

I. INTRODUCTION

The detection and classification of road signs and traffic lights play a crucial role in ensuring the safety and efficiency of autonomous driving systems. This project explores the application of computer vision techniques to detect and classify road signs and traffic lights in driving scenes. The primary objectives are to identify all relevant traffic signs within an image, determine their categories, and assess the status of traffic lights. To achieve these goals, we implemented a two-stage pipeline: detection of objects (road signs and traffic lights) and subsequent classification. The project is developed in Python using mainly the **OpenCV library**; other libraries are used.

This report outlines the development and evaluation of the proposed methodologies, highlighting their strengths and limitations. It also includes a comparative analysis of performance metrics, enabling a comprehensive understanding of the trade-offs between accuracy, efficiency, and scalability.

II. OBJECT DETECTION

The process that leads to the recognition of traffic signs and traffic lights consists of an initial segmentation phase, which divides the image into its most significant parts, followed by a classification phase of the segmented region that identifies the type of the object. These can be performed using various techniques, including the following:

- Deep Learning detection eg. YOLO
- *Classical* Machine Learning eg. Bag of Words
- *Deterministic* strategies eg. Hough transform

A. YOLO classifier

A **YOLO (You Only Look Once)** framework is a deep learning model specifically designed for real-time object detection and classification. We decided to use two different YOLO-based models: one pre-trained on the COCO (Common Objects in Context) dataset, and a custom-trained model tailored for detecting speed limit signs. The pretrained model, which is capable of detecting only traffic lights and stop signs, leverages the extensive COCO dataset to ensure robust detection of these classes. For this model, we opt for the latest version of YOLO provided by Ultralytics [2], a company specializing in AI-driven computer vision solutions. Specifically, we used the **YOLOv11x model**, which is their flagship version featuring the highest number of parameters (56.9 million). While this model incurs significant computational costs, the absence of strict real-time requirements in our application made execution time a secondary concern. For scenarios demanding real-time performance, lighter models, such as YOLOv11n or YOLOv11s, can be employed as alternatives.

The custom YOLO model, on the other hand, was trained using a specialized dataset, referenced in [1], consisting of 4,969 samples. This dataset enabled the training of a model capable of recognizing a wide variety of traffic-related classes, including: Green Light, Red Light, and various speed limits (10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120), as well as Stop signs. This comprehensive class structure ensures that the custom model is well suited for detecting and classifying speed limit signs across diverse conditions.

In the results section, we will analyse the performance and accuracy of this custom-trained model, comparing the outcomes achieved by training with different base models and varying numbers of epochs. These evaluations aim to

highlight the trade-off between computational efficiency and detection precision, as well as to validate the suitability of the custom model for traffic sign recognition tasks.

B. Hough transform

The **Hough Transform** is a feature extraction technique widely used for detecting shapes such as lines, circles, and ellipses within an image. While it boasts significant advantages, including high accuracy, robustness to noise, and versatility, it also comes with several limitations that must be carefully managed during its application. One notable drawback is its reliance on parameter tuning, which prevents universal detection. To achieve generalizable parameters, numerous trial-and-error attempts are often required, making the process time-consuming and less efficient. Additionally, the Hough Transform is not scale-invariant; significant changes in image resolution may necessitate manual adjustments or preprocessing to redefine the radius range, which can complicate the process.
Another limitation is its assumption of fixed-radius circles. If a circle in the image deviates from a perfect shape, the transform may fail to detect it altogether. Furthermore, the quality of edge detection and the contrast between the circle and its background play a critical role in successful detection. Poor edges or low contrast can severely impact results, increasing the risk of false negatives or inaccurate detections.

To improve the reliability of circle detection using the Hough Transform, preprocessing steps can greatly enhance its effectiveness. For instance, converting the image to greyscale and applying histogram equalization can improve contrast, while Gaussian de-noising filters help reduce noise that might interfere with edge detection. Moreover, techniques such as sky segmentation can eliminate irrelevant areas of the image, reducing false positives and focusing the detection on regions where circles are more likely to appear. These optimizations significantly enhance the precision and reliability of the Hough Transform, making it more suitable for practical applications.

C. Separation of ceil

Feature extraction techniques analyse the entire image, often leading to inefficiencies and false positives, particularly in scenarios with complex or cluttered environments. To address this, a custom function was developed to optimize deterministic algorithms by isolating the sky from the ground.

The `separate_image_by_brightness` function achieves this by dividing the image vertically based on the brightness distribution. It calculates the average brightness for each row and identifies a separation line where the cumulative brightness reaches 50% of the total brightness. This process effectively segments the image into two distinct regions: an upper region, typically brighter and corresponding to the sky, and a lower region, generally darker and associated with the ground.

This segmentation technique significantly improves the efficiency and accuracy of feature extraction by limiting the analysis to the region of interest, where road signs are most likely to appear. It minimizes the influence of irrelevant features in the sky, such as clouds or overexposed areas, which often cause false positives in detection. Furthermore, by focusing on the ground region, this method reduces the impact of environmental factors such as shadows, reflections, or uneven lighting, which can otherwise complicate the identification process. By isolating these areas we can enhances the robustness of deterministic algorithms.

In the code A the process is implemented using the `np.cumsum` function, which calculates the cumulative sum of the average brightness for each row of the image. Then a sorted search is performed to find the separation line corresponding to the threshold of 50%.

III. Object classification

A. Speed Limit Signs

The function `get_speed_limit`, detailed in the code snippet A, is designed to extract the speed limit value from an image of a Region of Interest (ROI) by utilizing **Optical Character Recognition (OCR)** capabilities provided by the *easyocr* library. This function identifies speed limit numbers on traffic signs by focusing on numerical data extraction and applying targeted preprocessing techniques.

To enhance OCR performance, the ROI image is first converted to greyscale, simplifying the colour information and emphasizing intensity differences that are critical for text recognition. A light Gaussian blur, with a kernel size of 5×5 and $\sigma_x = 1$, is then applied to reduce image noise while preserving the shape of the numbers, improving the overall readability of the text. This step ensures that artifacts or minor irregularities in the image do not interfere with the OCR's reading.

The OCR process itself is fine-tuned to recognize only numeric characters by using the parameter `allowlist='0123456789'`. This restriction narrows the scope of detection, making the OCR operation both faster and more reliable, as it eliminates unnecessary computations involving non-numeric characters. Once the OCR returns its results, the detected text is analysed iteratively to extract numeric values. Regular expressions are used for this task, specifically the pattern `(\b\d{1,3}\b)`, which identifies numbers consisting of 1 to 3 digits. This approach ensures flexibility in detecting common speed limit values while avoiding extraneous text that may inadvertently be captured by the OCR.

B. Traffic lights classification

The module processes the region of interest (ROI) by dividing it into three equal zones—upper, middle, and lower—which correspond to the typical positions of the red, yellow, and green signals. This division dynamically adapts to the orientation of the traffic light, whether it is vertical or horizontal. To analyse these zones, the ROI is first converted to greyscale, reducing the analysis to brightness levels. This eliminates dependency on colour, which is particularly advantageous when dealing with traffic lights featuring

coloured plastic covers or exposed to inconsistent lighting conditions. By focusing solely on pixel intensity, the system is resilient to variations that often compromise traditional methods.

The greyscale analysis calculates the average brightness of each zone and identifies the brightest one. To ensure accuracy, the module incorporates a variance check across the three zones. If the brightness variance is below a certain threshold, indicating uniform brightness, the system classifies the traffic light state as ”Unknown.” This step avoids false positives in scenarios where external factors, such as reflections or ambient lighting, might create uniform brightness across the entire ROI. Additionally, the module uses a dynamic threshold derived from the average brightness of the zones to determine whether the brightest zone is sufficiently distinct from the others to be classified as the active signal.

Compared to previous methods that relied on colour detection using HSV space, this brightness-based approach offers significant improvements. The older method, while effective in ideal conditions, was prone to inaccuracies when the traffic light's colours were obscured by tinted plastic or influenced by challenging lighting environments. These limitations often led to false detections, particularly in traffic lights with unconventional designs or when the background introduced colour noise. The new method eliminates these issues by entirely removing colour dependency, focusing instead on light intensity as the primary feature for classification. This shift enables the system to handle complex scenarios such as American-style traffic lights with coloured plastics, low-contrast environments, or variable illumination.

Another key advantage of the module is its ability to detect and handle the red-yellow combination, a feature common in countries like the UK. By analysing the brightness levels of the red and yellow zones simultaneously, the module can accurately identify this transitional state. Combined with its orientation-awareness and adaptive threshold, the system ensures robust performance across a wide range of traffic light configurations and environmental conditions.

IV. Results

This section is dedicated to comparing the various methods applied in this study, with a particular focus on the differences between deterministic approaches and those leveraging deep learning techniques. The analysis begins with an evaluation of the performance of the YOLO classifier, examining its capabilities in detecting and classifying traffic signs under various conditions. Following this, the traffic light detector is assessed, highlighting its effectiveness and limitations. Finally, the Hough Transform is analysed for its ability to detect road signs, particularly in scenarios involving simple geometric shapes.

The goal of this comparative analysis is to provide a comprehensive understanding of the strengths and weaknesses of each approach. While deterministic methods are often computationally efficient, they may lack the flexibility and accuracy required for complex real-world applications. In contrast, deep learning-based techniques, such as YOLO, offer superior performance in terms of precision and adaptability but come with higher computational costs.

Finally, the section concludes by drawing insights into which method proves to be the most effective for road sign and traffic light detection. This includes considerations of performance, computational efficiency, and adaptability to varying conditions, providing a clear recommendation on the optimal approach for these tasks.

A. YOLO models

As discussed in Section A, two models were employed for object recognition: a pre-trained model on the COCO dataset and a custom model. The custom model was trained using various base architectures and different numbers of epochs to evaluate performance across a range of complexities and training durations. Below is an overview of all the models generated, progressing from the lightest to the most complex:

- YOLO v8n 50 epochs
- YOLO v8n 100 epochs
- YOLO v8s 50 epochs
- YOLO 11s 50 epochs
- YOLO 11l 50 epochs

Among these, the YOLO 11l model is the most complex and computationally demanding. This model required the longest training time due to its significantly larger architecture and higher parameter count. Training it over 50 epochs provided the best balance between training duration and model performance for the specific dataset. The results obtained with the YOLO 11l model are demonstrated in the Confusion Matrix presented below, which offers a detailed analysis of the model's performance.

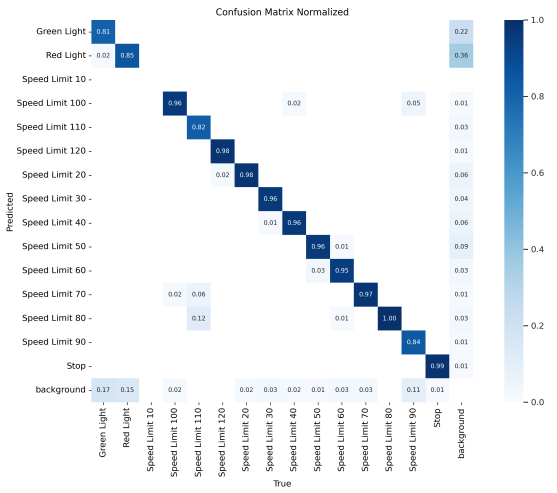

Fig. 1. Confusion Matrix Normalized

The matrix highlights the precision and recall rates across the various classes in the dataset, illustrating the model's ability to accurately detect and classify objects.

In many cases, the detection achieved by the YOLO

algorithm significantly outperformed traditional algorithms such as the Hough transform. To illustrate this, let us examine some examples that highlight the superior performance of YOLO in various scenarios.

Fig. 2. Custom trained YOLO detection of difficult road sign

Fig. 3. Custom trained YOLO detection of difficult road sign

These images showcase how YOLO effectively identifies objects with high accuracy, even in complex or challenging conditions where the Hough Transform tends to struggle.

B. Traffic Light Recognizer

To determine the state of a traffic light, several approaches were implemented and evaluated. The first approach did not include any spatial division of the traffic light's Region of Interest (ROI). Instead, color masks for the three typical traffic light colors—red, yellow, and green—were applied directly to the ROI, and the state was determined based on the most prominent color. However, this method exhibited the poorest performance, often producing false positives in scenarios where the traffic light was not perfectly captured or had significant noise. Examples of such cases are illustrated in the figure below.

Fig. 4. Detected status: Yellow
Sorted colours: `[(Yellow,11165), (Green,7484), (Red,0)]`

Fig. 5. Erroneous detection

These edge cases highlight a critical flaw: the lack of spatial segmentation within the traffic light's ROI leads to inaccurate state recognition. In the provided examples, background colours or overlapping elements introduce significant noise, making it difficult for the algorithm to discern the traffic light's actual state.

The second method introduces spatial segmentation, dividing the traffic light's ROI into three distinct zones corresponding to the positions of the red, yellow, and green lights. A colour mask is then applied to each zone independently, and the state of the traffic light is determined by identifying the most prominent colour in the respective zone. This approach significantly reduces false positives caused by background elements in the image and generally maintains good performance in recognizing the traffic light's state. Despite these improvements, the HSV color space's inherent characteristics pose challenges, particularly in detecting green within images with a marked greenish hue. In some cases, the overall green tone of the image disrupts the detection process, as shown in the examples below.

Fig. 6. Erroneous detection due to background tonality

From these figures, it is evident that when the image has a predominant green hue, the traffic light's state is misclassified, and "unknown" states are not correctly identified. This is a significant limitation when dealing with datasets where green tones are prevalent.

The final approach focuses solely on the spatial analysis of the three zones that make up the traffic light's ROI. The entire image is converted to grayscale, and the brightness levels of each region are analysed. To improve the detection of the "unknown" state, a condition based on the standard deviation (STD) of the brightness across the three zones was introduced. If the brightness levels in all three zones are approximately equal, the traffic light is categorized as "off" or "unknown." This method offers a highly accurate detection of the traffic light's state, although a few edge cases still result in errors. For instance, as shown in the figure below, when YOLO detects an excessively large ROI, the brightness variation across the three zones becomes minimal, leading to incorrect classification.

Fig. 7. Erroneous detection due to large ROI

While this approach achieves a high level of accuracy, its effectiveness is heavily reliant on the precision of the initial ROI detection. Any significant inaccuracies in the ROI, such as excessive size or misaligned regions, can still compromise the detection process.

C. Hough Transform

As discussed in Section II-B, the Hough Transform is not ideal for broad applications in detection tasks due to its limitations. One of the primary drawbacks is its sensitivity to image scaling, making it ineffective when objects appear at varying sizes in the image. Additionally, the transform struggles to detect circles when edges are irregular or poorly defined. Fine-tuning the parameters of the Hough Transform is a time-consuming process, and achieving consistent results across a large dataset of images can be particularly challenging.

To mitigate the risk of false positives, we implemented a validation step where Optical Character Recognition (OCR) serves as a necessary condition. If the text detected within a circle does not meet the expected criteria, the circle identified by the Hough Transform is discarded. This approach significantly reduces errors, particularly in complex scenarios.

Fig. 8. False detection of the Hough Transform

The sky segmentation technique described in Section II-C also helps to minimize false positives by effectively isolating the region of interest (ROI) containing road signs. This preprocessing step ensures that irrelevant areas, such as the sky, are excluded from the analysis, leading to more accurate detections. Some examples of successful detections using this method are shown in the figure below.

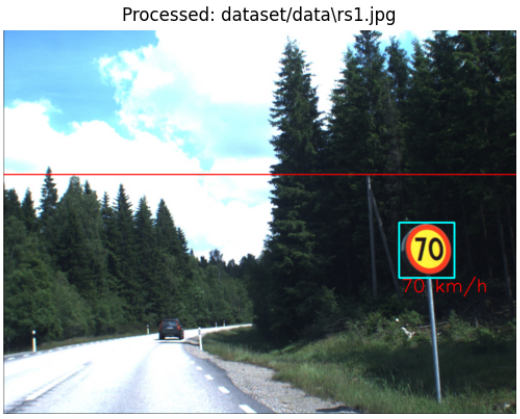

Fig. 9. False detection of the Hough Transform

Fig. 10. False detection of the Hough Transform

In these cases, it is evident that the sky separation line aligns well with the image content, preserving critical parts of the scene while eliminating unnecessary elements. This improves the overall accuracy of the Hough Transform in detecting road signs. However, there are instances where the detection fails to yield satisfactory results, as shown in the following figure.

Fig. 11. Erroneous detection from the OCR module, and no detection from the Hough transform on the right side

Fig. 12. No detection of the Hough Transform

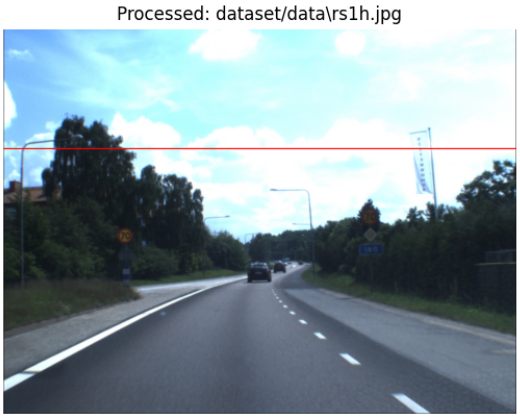

Fig. 13. No detection of the Hough Transform

In the images above, successful detection was only achieved using the YOLO deep learning model, which was specifically trained for road sign recognition. This highlights the limitations of the Hough Transform in handling complex or inconsistent image conditions, emphasizing the advantages of using more advanced techniques like YOLO for such tasks.

D. Hough vs YOLO

The YOLO detector demonstrates superior performance compared to the Hough Transform, particularly in challenging scenarios such as those depicted in Figures 8, 9, and 10. Even under these complex conditions, YOLO consistently detects road signs accurately and correctly identifies the speed limit values. This level of precision highlights YOLO's robustness and effectiveness in real-world applications, where factors such as partial occlusions, varying lighting conditions, and complex backgrounds can complicate detection tasks.

However, analysing images with the YOLO detector involves significant computational cost. The model's deep learning architecture requires substantial processing power, which may pose challenges for deployment in environments with limited resources or strict real-time constraints. Despite this, YOLO's greater flexibility in handling image scaling makes it a more adaptable solution. Unlike the Hough Transform, which struggles with scale invariance, YOLO can reliably detect objects across various image resolutions without the need for extensive parameter adjustments or preprocessing.

Given its higher accuracy and adaptability, YOLO is the preferred choice for road sign detection. Its ability to handle diverse image conditions and deliver consistent results outweighs computational demands, particularly in applications where detection precision and reliability are critical. This makes YOLO an ideal candidate for tasks such as autonomous driving, advanced driver assistance systems (ADAS), and traffic monitoring, where accuracy and robustness are paramount.

V. CONCLUSIONS

This project demonstrated the effectiveness of combining traditional methods like the Hough Transform with advanced deep learning models such as YOLO for detecting and classifying road signs and traffic lights. While deterministic approaches offered computational efficiency, their limitations in handling complex scenarios highlighted the superiority of deep learning techniques in terms of accuracy and adaptability. The integration of preprocessing steps, such as sky-ground segmentation and brightness-based ROI analysis, further improved detection reliability. These results underscore the critical role of advanced computer vision techniques in enabling robust autonomous driving systems. Future work could focus on optimizing computational performance and extending detection capabilities to handle more complex and dynamic environments.

REFERENCES

[1] Parisa Karimi Darabi (2024). *Traffic Signs Detection dataset for self-driving cars* Kaggle. https://www.kaggle.com/datasets/pkdarabi/cardetection

[2] *Ultralytics YOLO11*. 2025 Ultralytics Inc. https://docs.ultralytics.com/models/yolo11

[3] easyocr *API documentation* Jaided AI https://www.jaided.ai/easyocr/documentation/

[4] *OpenCV documentation* 2025 OpenCV team https://docs.opencv.org/4.x/index.html

[5] P. Zanuttigh, 2024 *Computer Vision course - Lecturer's slides*. University of Padua

APPENDIX A
PYTHON CODE

The analysis of the traffic light is performed in the next code frame, as described in III-B

```
def traffic_light_pos_color(ROI):
# convert the ROI to HSV color space
ROI = cv.cvtColor(ROI, cv.COLOR_BGR2HSV)

# split the image into three regions
height, width = ROI.shape[:2]

# split the image based on the orientation.
# Check if the image is vertical or horizontal
isVertical = height > width
if isVertical: # top - middle - bottom
r_region = ROI[0:int(height/3), :]
y_region = ROI[int(height/3):int(2*height/3), :]
g_region = ROI[int(2*height/3):, :]

else:   # left - middle - right
r_region = ROI[:, 0:int(width/3)]
y_region = ROI[:, int(width/3):int(2*width/3)]
g_region = ROI[:, int(2*width/3):]

# create masks for red, yellow, and green colours
mask_r = cv.inRange(r_region, lower_r, upper_r)
[...]

# count the number of non-zero pixels in each mask
r_count = cv.countNonZero(mask_r)
[...]

color_counts = {
    "Red": r_count,
    "Yellow": y_count,
    "Green": g_count
}

sorted_colors = sorted(color_counts.items(),
            key=lambda x: x[1], reverse=True)

# get the most prominent color
prominent_color, max_count = sorted_colors[0]

# set a threshold based on the average count
th = max(ROI.shape[0] * ROI.shape[1] * 0.0005,
    int(np.mean([r_count, y_count, g_count])))

# Check for red + yellow combination (British)
if (prominent_color == "Red" or
    prominent_color == "Yellow") and
    (red_count > th and yellow_count > th):
        return "Yellow with Red"

if max_count > th:
    return prominent_color
else:
    return "Unknown"
```

The next section of code perform the detection of the traffic light status, based on the intensity approach used

```
def traffic_light_pos_BW(ROI):
gray = cv.cvtColor(ROI, cv.COLOR_BGR2GRAY)

# split the image into three regions
height, width = gray.shape[:2]


isVertical = height > width
if isVertical:
r_region = gray[0:int(height/3), :]
y_region = gray[int(height/3):int(2*height/3), :]
g_region = gray[int(2*height/3):, :]

else:
r_region = gray[:, 0:int(width/3)]
y_region = gray[:, int(width/3):int(2*width/3)]
g_region = gray[:, int(2*width/3):]

# calculate the average pixel intensity
red_intensity = np.mean(red_region)
yellow_intensity = np.mean(yellow_region)
green_intensity = np.mean(green_region)

intensity_val = {
    "Red": red_intensity,
    "Yellow": yellow_intensity,
    "Green": green_intensity
}

if np.std([intensities]) < 15:
    return "Unknown"

sorted_intensities = sorted(intensity_val.items(),
key=lambda x: x[1], reverse=True)
prominent_color, max_count = sorted_intensities[0]

th = max(np.mean([intensities])*1.2, 60)

# Check for red + yellow combination (British)
if (prominent_color == "Red" or
prominent_color == "Yellow") and
(red_count > th and yellow_count > th):
    return "Yellow with Red"

if max_count > threshold:
    return prominent_color
else:
    return "Unknown"
```

The next section of code performs the separation of the sky from the ground as described in II-C.

```
def separate_image_by_brightness(image):
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (9, 9), 2)
height, width = gray.shape

# Calcola la luminosità media per ogni riga
row_bright = np.mean(gray, axis=1)

# Calcola la somma cumulativa della luminosità
cum_bright = np.cumsum(row_bright)
cum_bright_norm = cum_bright / cum_bright[-1]
```

```
# Trova la linea di separazione basata
# su un punto medio (e.g., 50% cumulativo)
sep_line = np.searchsorted(cum_bright_norm, 0.5)

ceil_zone = image[:sep_line, :]
ground_zone = image[sep_line:, :]

return ceil_zone, ground_zone, sep_line
```

The next section of code perform the hough transforms.

```
def detect_circles(image,dp, min_dist,param1,
param2, min_radius,max_radius) -> np.ndarray:
# Apply histogram equalization to improve contrast
image = histeq(image)
gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

# Apply GaussianBlur to reduce noise
blurred = cv.GaussianBlur(gray, (9, 9), 2)

# Detect circles using HoughCircles
circles = cv.HoughCircles(blurred,
cv.HOUGH_GRADIENT, dp, min_dist,
param1=param1, param2=param2,
minRadius=min_radius, maxRadius=max_radius)

return circles
```

The following section of code performs the detection of the speed limit value as described in III-A.

```
def get_speed_limit(ROI) -> int:
# convert the ROI to grayscale
gray = cv.cvtColor(ROI, cv.COLOR_BGR2GRAY)
# Apply preprocessing to enhance OCR performance
blur = cv.GaussianBlur(gray, (5, 5), 1)

# Use EasyOCR to extract text
reader = easyocr.Reader(['en'], gpu=True)
result = reader.readtext(blur, allowlist='0...9')

# Extract numbers from the detected text
for detection in result:
    text = detection[1]  # The detected text
    # Search for a number with 1-3 digits
    match = re.search(r'\b\d{1,3}\b', text)
    if match:
        # Return the first number found
        return int(match.group(0))
```

The following section of code performs entire detection with the various methods.

```

# Load the YOLO model
modelCOCO = YOLO("yolo11x.pt")
modelTRAIN = YOLO("best11s50.pt")

# Detect object in the image with YOLO
# Only class traffic light and stop sign allowed
resultsCOCO = modelCOCO.predict(image,imgsz=
image.shape[:2], classes=[9,11])
resultsTRAIN = modelTRAIN.predict(image)

# Detect circles in the image with HoughCircles
circles = detect_circles(
    image[separate_ceil(image):, :],
    dp, min_dist,param1,param2,
    min_radius,max_radius)

cv.line(image, (0, separate_ceil(image)),
(image.shape[1], separate_ceil(image)),
(0, 0, 255), 2)

detectionsCOCO = resultsCOCO[0].
boxes.data.cpu().numpy()  # YOLO detection results

for box in detectionsCOCO:
x1, y1, x2, y2, conf, cls = map(int, box)
label = resultsCOCO[0].names[cls]

# Traffic light analysis
if label == "traffic light":
    region = image[y1:y2, x1:x2]
    status = traffic_light_pos_BW(region)
    cv.putText(image, status, (x1, y2 + 20),
        cv.FONT_HERSHEY_SIMPLEX, 0.5,
            (255, 0, 0), 2)
    cv.rectangle(image, (x1, y1), (x2, y2),
        (0, 255, 0), 2)
    cv.putText(image, label, (x1, y1 - 10),
        cv.FONT_HERSHEY_SIMPLEX, 0.5,
            (0, 255, 0), 2)

# Stop signal
else:
    cv.rectangle(image, (x1, y1),
        (x2, y2), (0, 255, 0), 2)
    cv.putText(image, label, (x1, y1 - 10),
        cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

detectionsTRAIN = resultsTRAIN[0].
boxes.data.cpu().numpy()

for box in detectionsTRAIN:
x1, y1, x2, y2, conf, cls = map(int, box)
label = resultsTRAIN[0].names[cls]

cv.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
cv.putText(image, label, (x1, y1 + 10),
    cv.FONT_HERSHEY_SIMPLEX, 0.5, (255,0, 0), 2)

# HoughCircles detected circles
if circles is not None:
# Convert the circle parameters to integers
circles = np.uint16(np.around(circles))
for circle in circles[0, :]:
center = (circle[0], circle[1])  # x, y
center = (center[0],
    center[1] + separate_ceil(image))
radius = circle[2]

# Extract the ROI for speed limit sign
region = image[center[1] - radius -15:
    center[1] + radius +15,
    center[0] - radius -15: center[0] + radius+15]
# Get the speed limit from the ROI with OCR
speed_limit = get_speed_limit(region)
```

```
if speed_limit is not None:
    cv.putText(image, f"{speed_limit} km/h",
    (center[0] - radius, center[1] + radius + 40),
    cv.FONT_HERSHEY_SIMPLEX, 1.5, (0, 0, 255), 2)
    cv.rectangle(image, (center[0] - radius - 5,
    center[1] - radius - 5), (center[0] + radius +5,
    center[1] + radius+5), (255, 255, 0), 3)

return image
```