Project Report for

RE4012 - MACHINE VISION

Irish Speed-Signs Detection and Classification

Revision Timestamp: 04/04/2025 05:15:08

Project Team

Student Name	Student Id
ZEXIN LI	23202637
QINYUAN LIU	20137095

Contents

1. Introduction

This project aims to achieve automatic detection and classification of Irish speed limit signs (40, 50, 60, 80, 100, 120 km/h) in road scenes. Speed limit signs are key visual cues in road traffic and are of great significance for ensuring traffic safety and guiding driving behavior. With the development of autonomous driving and intelligent transportation systems, how to quickly and accurately identify speed limit signs in images has become a practical application problem in computer vision.

This system adopts a **two-stage architecture** to realize speed limit sign recognition: first, the red area is extracted based on the **HSV color space**, and the candidate **region of interest** (RoI) is screened out through morphological processing and connected domain analysis; then each **RoI is converted into a 4096-dimensional unit vector**, and the **1-NN classifier** is used to compare with the template library to complete the classification and confirmation of the speed value.

2. Approach

The project is undertaking by 2 individuals, with attempt of recruit new members made. Unfortunately, we have not yet received a reply from suitable candidates. The project is implemented in 2 phases: Code development and Report writing. Zexin Li has undertaken the main part of the code development, and Qinyuan Liu has taken responsibility for code review and Report writing.

3. Workflow of the Code

Based on the idea of the project guide PDF file, this project is designed as a clear two-stage process: first, the region proposal module (RoI Finder) quickly selects red areas that may contain speed limit signs from the entire image, and then the region classification module (RoI Classifier) performs more detailed identification and judgment on these candidate areas.

In the first stage, we convert the image to the HSV color space and set the threshold with hue close to red (about 0 or 1), high saturation and moderate brightness to generate a preliminary red mask. Then, morphological operations (such as dilation and erosion) are used to repair the broken areas, and all red blocks are extracted through connected domain analysis. For areas with too small area or too large aspect ratio, we directly discard them - because most real speed limit signs are approximately circular and have a certain degree of significance.

In the second stage, we crop the image fragments of each retained area (i.e. RoI), uniformly scale them to 64×64 pixels and convert them to grayscale images, and then perform contrast stretching and normalization to make them 4096-dimensional unit vectors. These vectors are then compared one by one with the feature vectors in the template library (i.e., existing speed limit sign examples), and the 1-NN classifier is used to find the nearest neighbor. If the shortest distance exceeds the set threshold (such as 1.2) or matches a non-sign sample (labeled as -1), the area is judged invalid.

Finally, we frame all successfully identified speed limit signs in the original image with red rectangles and mark their corresponding speed values; at the same time, debug information is output in the terminal window, including the area screening process, classification labels, and confidence judgments. This process is not only clearly structured and module-independent, but also convenient for future migration to other types of signs or replacement of more advanced classifier architectures (such as SVM or CNN)

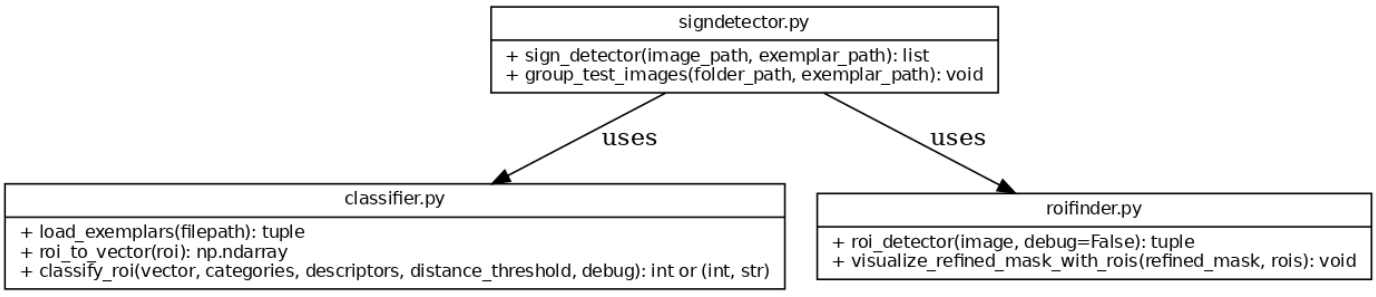


Figure 1 Project UML Relational Diagram

4. Code Implementation:

1. Image pre-processing and mask construction (Hue/Sat/Val)

First, convert the input image into HSV space, set thresholds on the three channels of Hue (hue), Saturation (saturation), and Value (brightness), and construct a mask to extract the red area. The intersection of the three forms a relatively clean candidate area map (binary_mask), the purpose of which is to locate the area in the image that may contain the red circle speed mark.

```
hue_mask = (hsv[:, :, 0] > 0.95) | (hsv[:, :, 0] < 0.09)
sat_mask = hsv[:, :, 1] > 0.4
val_mask = hsv[:, :, 2] > 0.15

binary_mask = hue_mask & sat_mask & val_mask
```

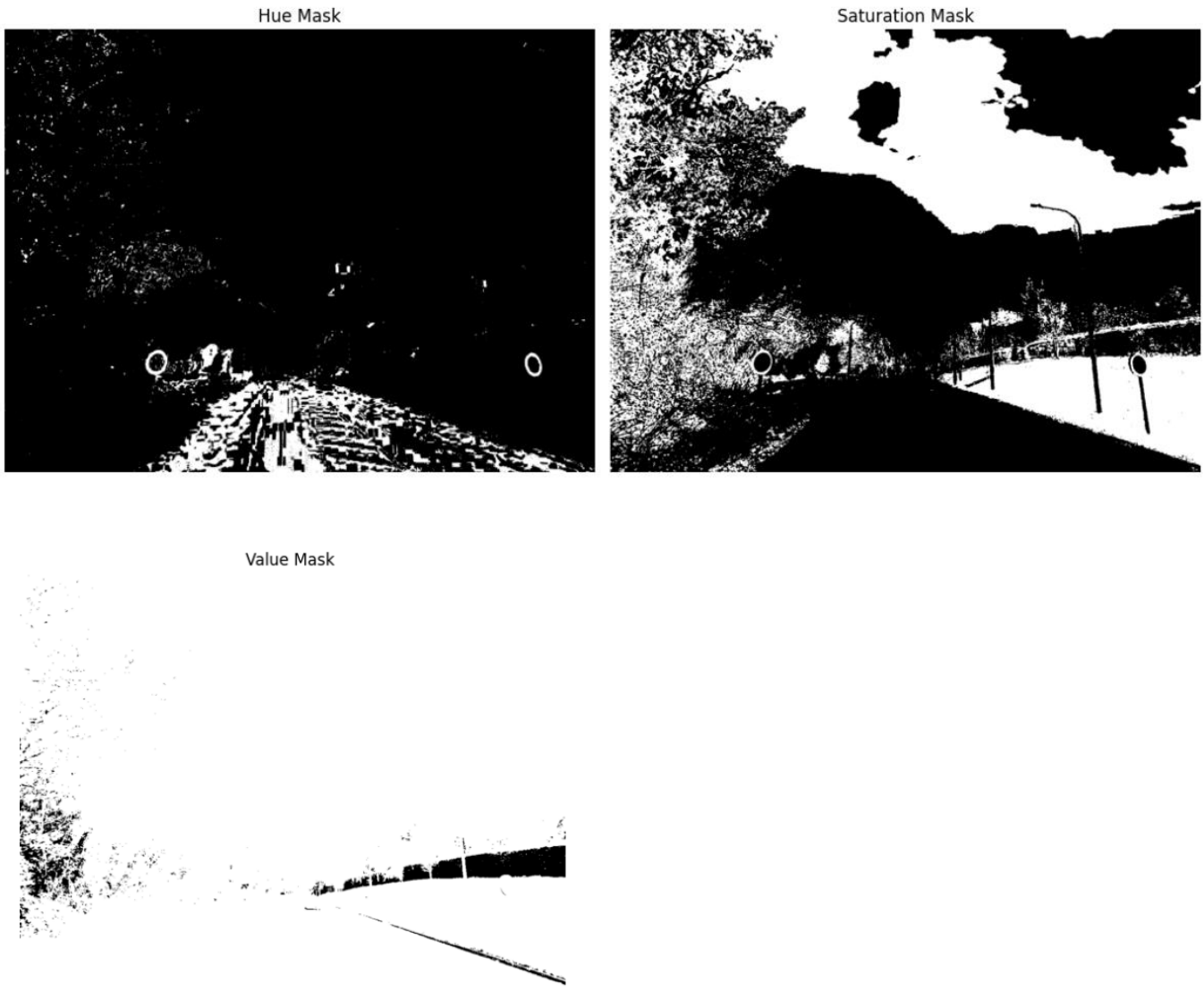


Figure 2 Image pre-processing and mask construction

2. Morphological processing and noise suppression (Refined Mask)

The candidate mask image is processed by binary_opening and binary_closing to form a refined_mask. In this stage, isolated small noise points are cleaned up and broken areas are completed to obtain a more complete connected graph, which is convenient for subsequent regional analysis.

```
selem = morphology.square(5)
    selem = morphology.disk(1)
    opened_mask = morphology.binary_opening(binary_mask, selem)
    refined_mask = morphology.binary_closing(opened_mask, selem)
```

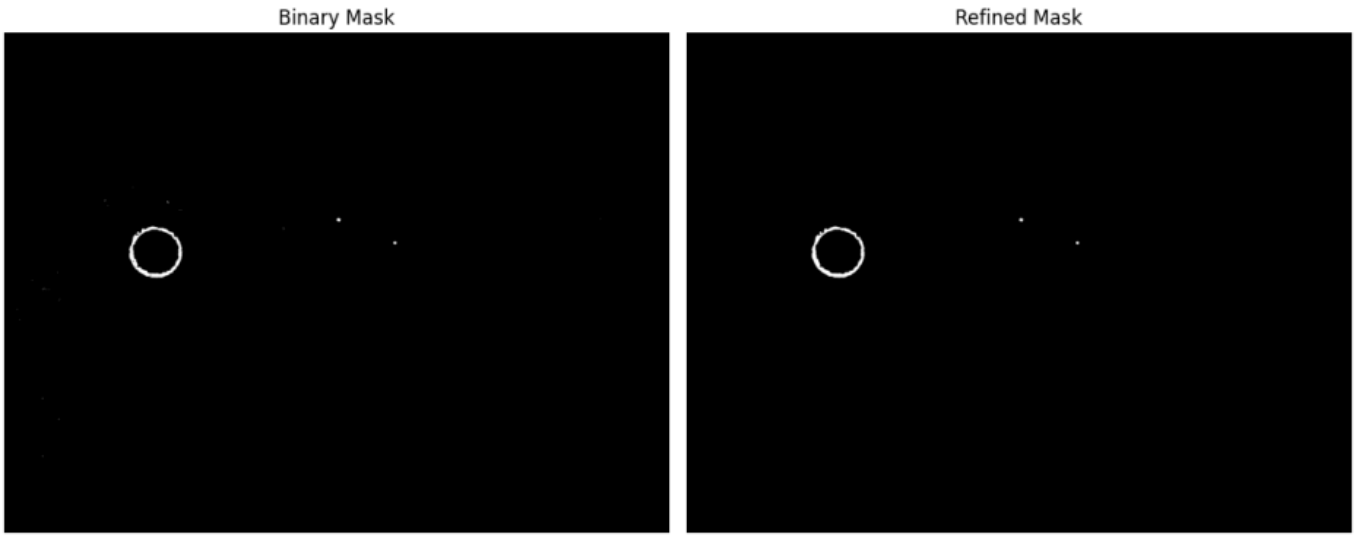


Figure 3 Morphological processing and noise suppression

3. RoI extraction and filtering (region proposal)

Next, the system performs a connected region analysis on the refined_mask and calculates the area and aspect ratio of each connected region. Only regions with an area > 280 and a reasonable aspect ratio (close to a circle) are considered "candidate RoIs", that is, image segments that may be real speed limit signs. This screening result is directly recorded in the rois list.

```
    labeled_mask, num_features = label(refined_mask)
    slices = find_objects(labeled_mask)

    rois = []
    rejection_info = []
    for i, slc in enumerate(slices):
        if slc is None:
            continue
        region_mask = (labeled_mask[slc] == (i + 1))
        area = np.sum(region_mask)
        info_str = f"Region {i}: area = {area}. "
        passed = True

        if area <= 280:
            info_str += "Rejected: area too small (<=280)."
            passed = False
        else:
            minr, maxr = slc[0].start, slc[0].stop
            minc, maxc = slc[1].start, slc[1].stop
            height = maxr - minr
            width = maxc - minc
            ratio = width / height
            info_str += f"Width/Height = {ratio:.2f}. "

            if ratio >= 1.8:
                info_str += "Rejected: width/height ratio too high (>=1.8)."
                passed = False

        if passed:
            minr, maxr = slc[0].start, slc[0].stop
            minc, maxc = slc[1].start, slc[1].stop
            rois.append((minr, minc, maxr, maxc))
            info_str += "Accepted."
        rejection_info.append(info_str)

    if debug:
        return rois, binary_mask, opened_mask, refined_mask, hue_mask, sat_mask,
val_mask, rejection_info
    else:
        return rois, binary_mask, opened_mask, refined_mask, hue_mask, sat_mask, val_mask
```

4. Image vectorization and normalization (RoI Vector)

For each RoI image segment, perform the following processing:

a. Convert to grayscale

b. Scale to 64×64 pixels

c. Histogram equalization to enhance contrast

d. Flatten to 4096-dimensional vector

Normalize to unit vector after zero mean processing

These processing are completed by the roi_to_vector function, the purpose is to normalize the image into a feature space representation that is easy to compare.

```
def roi_to_vector(roi):
    if roi.ndim == 3 and roi.shape[-1] == 4:
        roi = roi[..., :3]
    gray = color.rgb2gray(roi)
    resized = transform.resize(gray, (64, 64), anti_aliasing=True)
    enhanced = exposure.equalize_adapthist(resized)
    vector = enhanced.flatten()
    vector = vector - np.mean(vector)
    norm = np.linalg.norm(vector)
    if norm > 1e-12:
        vector /= norm
    return vector
```

5. 1-NN classification and template comparison (Classification)

Each feature vector is Euclidean distance calculated with the template in the exemplar database to find the template with the closest distance (1-NN).

If the label of the nearest neighbor is -1, or the distance exceeds the threshold of 1.2, the RoI is judged as a non-mark;

Otherwise, the corresponding speed label (such as 50, 60, 120) is returned as the recognition result.

```
def classify_roi(vector, categories, descriptors, distance_threshold=1.2, debug=False):
    distances = np.linalg.norm(descriptors - vector, axis=1)
    nearest_index = np.argmin(distances)
    min_dist = distances[nearest_index]
    predicted_label = categories[nearest_index]
    debug_info = f"Min distance: {min_dist:.3f}. "

    if predicted_label == -1:
        debug_info += "Matched negative sample (-1)."
        return (-1, debug_info) if debug else -1

    if min_dist > distance_threshold:
```

```
        debug_info += f"Distance {min_dist:.3f} exceeds threshold {distance_threshold}."
        return (-1, debug_info) if debug else -1

    debug_info += "Accepted."
    return (predicted_label, debug_info) if debug else predicted_label
```

6. Final visualization and terminal output

All successfully identified RoIs will be drawn with red boxes on the original image, with the detected speed values marked, and debugging information such as region coordinates, matching distance, and identification labels will be output to the terminal. All of this is integrated in the sign_detector() function.

```
        rect = patches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                 fill=False, edgecolor='red', linewidth=2)
        ax.add_patch(rect)
        ax.text(minc, minr, f"Speed: {label}", color='white',
                fontsize=12, verticalalignment='top')

    ax.set_title('Detected Regions with 1-NN Classification')
    ax.axis('off')
    plt.show()
```

Figure 4 Detected Regions with 1-NN Classification

5. Conclusion

This project fully realizes the automatic detection and classification process of speed limit signs in Ireland. Through **HSV color space analysis, morphological processing** and **connected domain analysis**, the potential sign area is accurately extracted, and the 1-NN classifier is used to compare with the template feature vector to achieve accurate recognition and visual output of speed information. The entire system has a clear structure and clear layers, which fully reflects the integrated processing flow of image preprocessing, feature extraction, classification judgment and result presentation. It has certain similarities with the medical image reconstruction project "Image Reconstruction from a Sinogram" in process structure, but this project focuses more on the detection and recognition of real-life targets, demonstrating the effective combination of color space processing, image segmentation and traditional pattern recognition methods. This method has good interpretability and scalability, and is suitable for a wider range of traffic scene target recognition tasks.

6. References

[1] C. Flanagan, *Irish Speed-Signs Detection and Classification*, RE4012/7 Machine Vision Project 2, University of Limerick, 2024–2025.

[2] Scikit-Image Documentation. [Online]. Available: https://scikit-image.org/

[3] NumPy Documentation. [Online]. Available: https://numpy.org/doc/

7. Appendix A: Results

The following pages are output of the Testing Results.

singdetector

April 4, 2025

[15]:
```
import os
import time
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from skimage import io

from classifier import load_exemplars, roi_to_vector, classify_roi
from roifinder import roi_detector, visualize_refined_mask_with_rois

#  Jupyter
%matplotlib inline

def sign_detector(image_path, exemplar_path):
    image = io.imread(image_path)

    rois, binary_mask, opened_mask, refined_mask, hue_mask, sat_mask, val_mask,␣
 ↪rejection_info = roi_detector(image, debug=True)

    for info in rejection_info:
        print(info)

    fig, axes = plt.subplots(2, 3, figsize=(18, 12))
    ax = axes.ravel()

    ax[0].imshow(image)
    ax[0].set_title('Original Image')
    ax[0].axis('off')

    ax[1].imshow(hue_mask, cmap='gray')
    ax[1].set_title('Hue Mask')
    ax[1].axis('off')

    ax[2].imshow(sat_mask, cmap='gray')
    ax[2].set_title('Saturation Mask')
    ax[2].axis('off')

    ax[3].imshow(val_mask, cmap='gray')
```

```
    ax[3].set_title('Value Mask')
    ax[3].axis('off')

    ax[4].imshow(binary_mask, cmap='gray')
    ax[4].set_title('Binary Mask')
    ax[4].axis('off')

    ax[5].imshow(refined_mask, cmap='gray')
    ax[5].set_title('Refined Mask')
    ax[5].axis('off')

    plt.tight_layout()
    plt.show()

    categories, descriptors = load_exemplars(exemplar_path)
    detected_speeds = []

    for roi in rois:
        roi_img = image[roi[0]:roi[2], roi[1]:roi[3]]
        vector = roi_to_vector(roi_img)
        label, debug_msg = classify_roi(vector, categories, descriptors,␣
↪distance_threshold=1.2, debug=True)
        print(f"ROI at {roi} classified as {label}. {debug_msg}")

    fig, ax = plt.subplots(figsize=(10, 8))
    ax.imshow(image)
    for (minr, minc, maxr, maxc) in rois:
        roi_img = image[minr:maxr, minc:maxc]
        roi_vec = roi_to_vector(roi_img)
        label = classify_roi(roi_vec, categories, descriptors)

        if label == -1:
            continue

        detected_speeds.append(label)

        rect = patches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                 fill=False, edgecolor='red', linewidth=2)
        ax.add_patch(rect)
        ax.text(minc, minr, f"Speed: {label}", color='white',
                fontsize=12, verticalalignment='top')

    ax.set_title('Detected Regions with 1-NN Classification')
    ax.axis('off')
    plt.show()

    return detected_speeds
```

```
def group_test_images(folder_path, exemplar_path, group_number):
    """
       group_number    input()
    """
    groups = {
        1: ["40-0001x1.png", "40-0002x1.png"],
        2: ["50-0001x1.png", "50-0002x1.png", "50-0003x1.png", "50-0004x2.png",␣
↪"50-0005x1.png"],
        3: ["60-0001x1.png", "60-0002x1.png", "60-0003x1.png", "60-0004x2.png",␣
↪"60-0005x1.png"],
        4: ["80-0001x1.png", "80-0002x2.png", "80-0003x2.png", "80-0004x2.png",␣
↪"80-0005x1.png"],
        5: ["100-0001x1.png", "100-0002x1.png", "100-0003x1.png", "100-0004x2.
↪png", "100-0005x1.png"],
        6: ["120-0001x1.png", "120-0002x2.png", "120-0003x1.png", "120-0004x1.
↪png", "120-0005x1.png"],
        7: ["50-0002x1.png", "80-0003x2.png"]
    }

    if group_number not in groups:
        print(f"Group {group_number} not defined. Available groups are:␣
↪{list(groups.keys())}")
        return

    images_to_test = groups[group_number]
    print(f"\nNow testing group {group_number} with images: {images_to_test}")

    for img in images_to_test:
        full_path = os.path.join(folder_path, img)
        if not os.path.exists(full_path):
            print(f"File not found: {full_path}")
            continue

        print(f"\nProcessing {img} ...")
        detected_speeds = sign_detector(full_path, exemplar_path)
        count = len(detected_speeds)
        if count == 0:
            print("No speed signs detected.")
        elif count == 1:
            print(f"Detected 1 speed sign: {detected_speeds[0]} km/h")
        else:
            speeds_str = ", ".join([f"{s} km/h" for s in detected_speeds])
            print(f"Detected {count} speed signs: {speeds_str}")
```

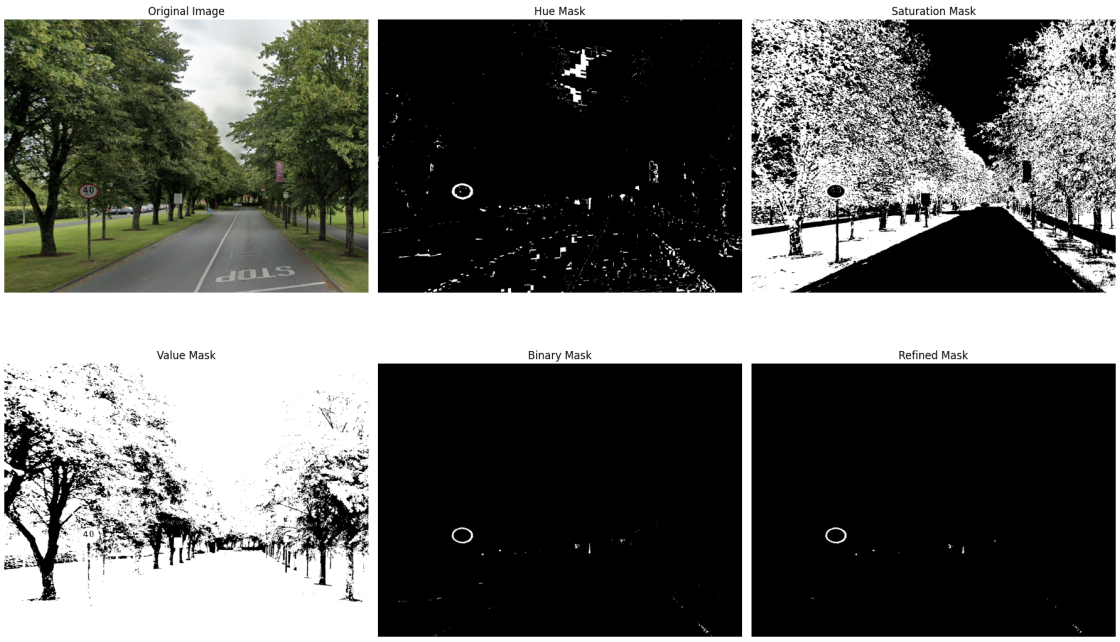
```
        plt.close('all')
        time.sleep(1)
```

[17]:
```
#    group
def group_test_images(folder_path, exemplar_path, group_number):
    groups = {
        1: ["40-0001x1.png", "40-0002x1.png"],
        2: ["50-0001x1.png", "50-0002x1.png", "50-0003x1.png", "50-0004x2.png",␣
 ↪"50-0005x1.png"],
        3: ["60-0001x1.png", "60-0002x1.png", "60-0003x1.png", "60-0004x2.png",␣
 ↪"60-0005x1.png"],
        4: ["80-0001x1.png", "80-0002x2.png", "80-0003x2.png", "80-0004x2.png",␣
 ↪"80-0005x1.png"],
        5: ["100-0001x1.png", "100-0002x1.png", "100-0003x1.png", "100-0004x2.
 ↪png", "100-0005x1.png"],
        6: ["120-0001x1.png", "120-0002x2.png", "120-0003x1.png", "120-0004x1.
 ↪png", "120-0005x1.png"],
        7: ["50-0002x1.png", "80-0003x2.png"]
    }

    if group_number not in groups:
        print(f"Group {group_number} not defined.")
        return

    images_to_test = groups[group_number]
    print(f"\n[Group {group_number}] Testing images: {images_to_test}")

    for img in images_to_test:
        full_path = os.path.join(folder_path, img)
        print(f"→ Now processing: {full_path}")

        if not os.path.exists(full_path):
            print(f"File not found: {full_path}")
            continue

        try:
            detected_speeds = sign_detector(full_path, exemplar_path)
            count = len(detected_speeds)
            if count == 0:
                print("No speed signs detected.")
            elif count == 1:
                print(f"Detected 1 speed sign: {detected_speeds[0]} km/h")
            else:
                speeds_str = ", ".join([f"{s} km/h" for s in detected_speeds])
                print(f"Detected {count} speed signs: {speeds_str}")

        except Exception as e:
```

```
            print(f"Error processing {img}: {e}")

        plt.close('all')
        time.sleep(1)

        #    group
def test_all_groups(folder_path, exemplar_path):
    for group_number in range(1, 8):  # group 1~7
        print(f"\n===== Testing Group {group_number} =====")
        group_test_images(folder_path, exemplar_path, group_number)
```

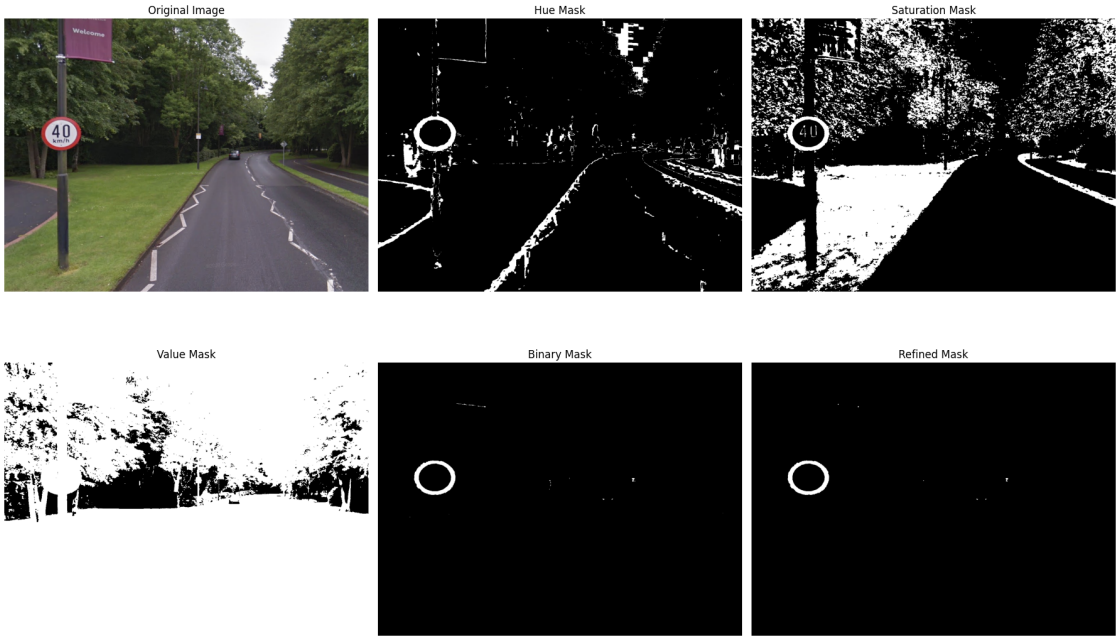
[ ]:
```
test_all_groups("image", "1-NN-descriptor-vects.npy")
```

```
===== Testing Group 1 =====

[Group 1] Testing images: ['40-0001x1.png', '40-0002x1.png']
→ Now processing: image\40-0001x1.png
Region 0: area = 1095. Width/Height = 1.32. Accepted.
Region 1: area = 18. Rejected: area too small (<=280).
Region 2: area = 25. Rejected: area too small (<=280).
Region 3: area = 54. Rejected: area too small (<=280).
Region 4: area = 5. Rejected: area too small (<=280).
Region 5: area = 105. Rejected: area too small (<=280).
Region 6: area = 31. Rejected: area too small (<=280).
Region 7: area = 5. Rejected: area too small (<=280).
Region 8: area = 29. Rejected: area too small (<=280).
Region 9: area = 5. Rejected: area too small (<=280).
Region 10: area = 8. Rejected: area too small (<=280).
Region 11: area = 5. Rejected: area too small (<=280).
Region 12: area = 8. Rejected: area too small (<=280).
Region 13: area = 11. Rejected: area too small (<=280).
Region 14: area = 33. Rejected: area too small (<=280).
Region 15: area = 10. Rejected: area too small (<=280).
Region 16: area = 8. Rejected: area too small (<=280).
Region 17: area = 26. Rejected: area too small (<=280).
```




```
ROI at (576, 259, 633, 334) classified as 40.0. Min distance: 0.473. Accepted.
```

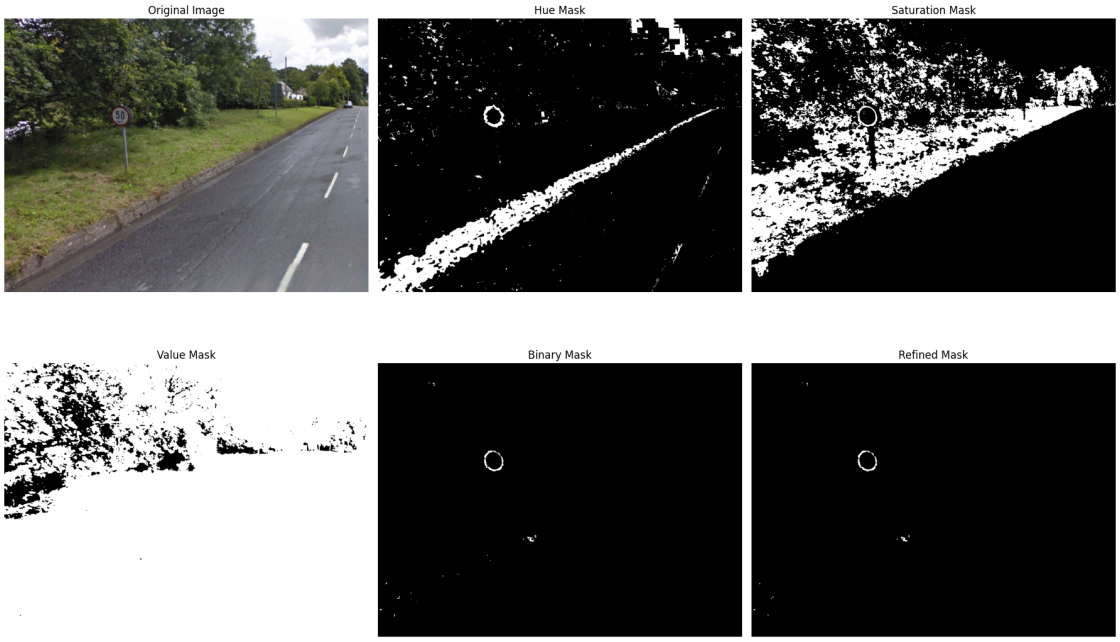
```
Detected 1 speed sign: 40.0 km/h
→ Now processing: image\40-0002x1.png
Region 0: area = 8. Rejected: area too small (<=280).
Region 1: area = 12. Rejected: area too small (<=280).
Region 2: area = 5242. Width/Height = 1.21. Accepted.
Region 3: area = 63. Rejected: area too small (<=280).
Region 4: area = 5. Rejected: area too small (<=280).
Region 5: area = 14. Rejected: area too small (<=280).
Region 6: area = 10. Rejected: area too small (<=280).
```




```
ROI at (344, 127, 465, 273) classified as 40.0. Min distance: 0.380. Accepted.
```

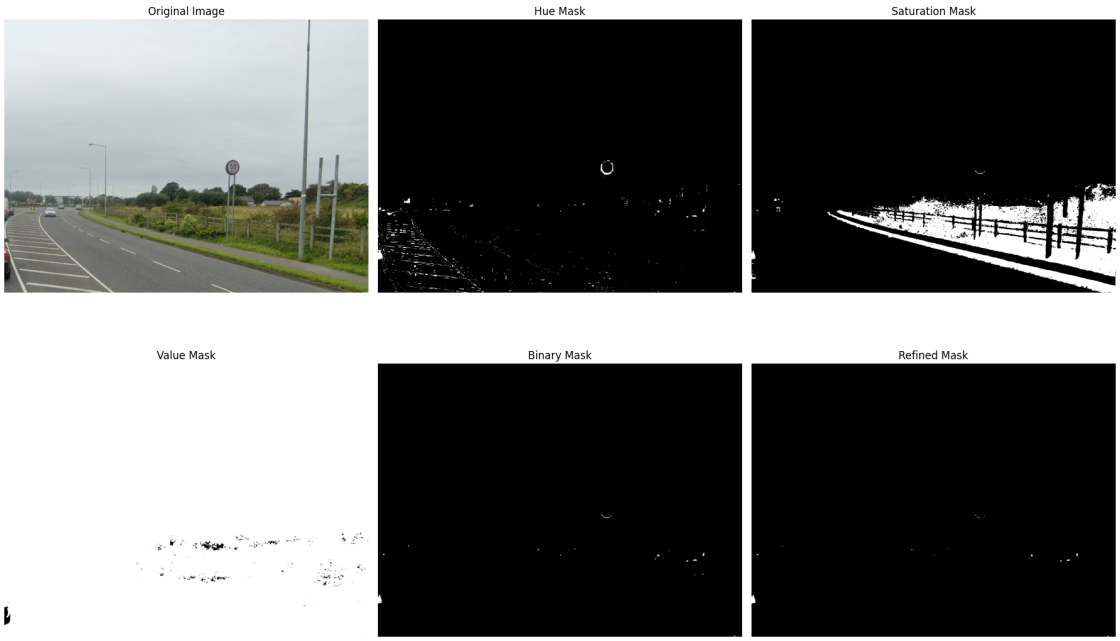

```
Detected 1 speed sign: 40.0 km/h

===== Testing Group 2 =====

[Group 2] Testing images: ['50-0001x1.png', '50-0002x1.png', '50-0003x1.png',
'50-0004x2.png', '50-0005x1.png']
→ Now processing: image\50-0001x1.png
Region 0: area = 5. Rejected: area too small (<=280).
Region 1: area = 16. Rejected: area too small (<=280).
Region 2: area = 8. Rejected: area too small (<=280).
Region 3: area = 295. Width/Height = 1.57. Accepted.
Region 4: area = 48. Rejected: area too small (<=280).
Region 5: area = 717. Width/Height = 1.28. Accepted.
Region 6: area = 26. Rejected: area too small (<=280).
Region 7: area = 173. Rejected: area too small (<=280).
Region 8: area = 10. Rejected: area too small (<=280).
Region 9: area = 10. Rejected: area too small (<=280).
Region 10: area = 11. Rejected: area too small (<=280).
Region 11: area = 10. Rejected: area too small (<=280).
Region 12: area = 21. Rejected: area too small (<=280).
```

Region 13: area = 5. Rejected: area too small (<=280).
Region 14: area = 5. Rejected: area too small (<=280).



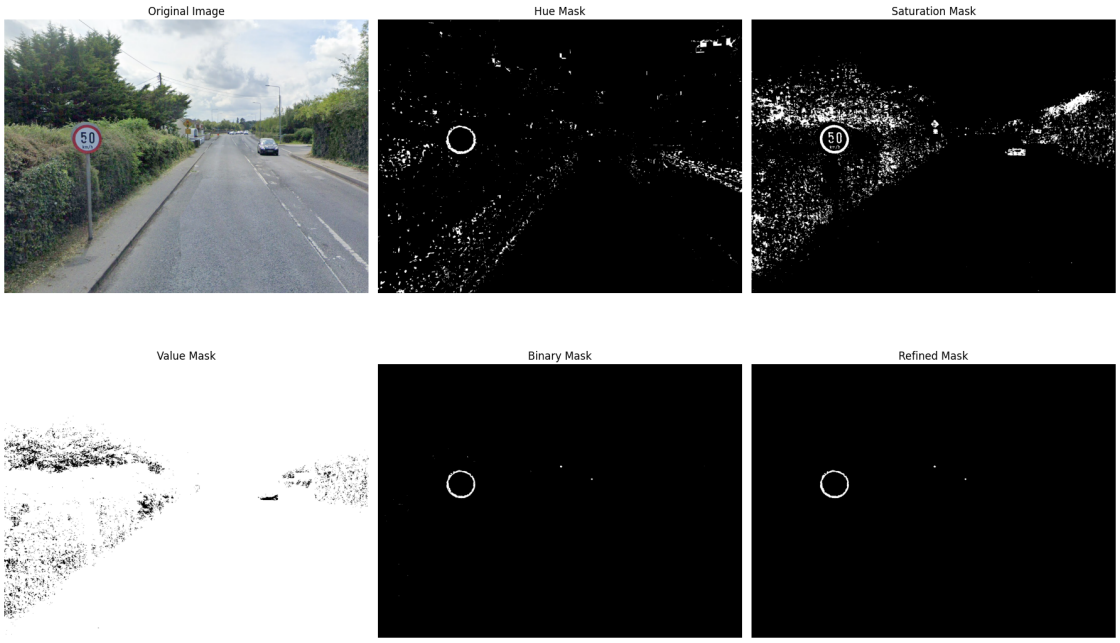
ROI at (307, 374, 335, 418) classified as -1. Min distance: 1.210. Distance 1.210 exceeds threshold 1.2.
ROI at (326, 373, 380, 442) classified as 50.0. Min distance: 1.081. Accepted.


```
Detected 1 speed sign: 50.0 km/h
→ Now processing: image\50-0002x1.png
Region 0: area = 8. Rejected: area too small (<=280).
Region 1: area = 5. Rejected: area too small (<=280).
Region 2: area = 10. Rejected: area too small (<=280).
Region 3: area = 5. Rejected: area too small (<=280).
Region 4: area = 5. Rejected: area too small (<=280).
Region 5: area = 5. Rejected: area too small (<=280).
Region 6: area = 12. Rejected: area too small (<=280).
Region 7: area = 41. Rejected: area too small (<=280).
Region 8: area = 20. Rejected: area too small (<=280).
Region 9: area = 34. Rejected: area too small (<=280).
Region 10: area = 5. Rejected: area too small (<=280).
Region 11: area = 5. Rejected: area too small (<=280).
Region 12: area = 44. Rejected: area too small (<=280).
Region 13: area = 20. Rejected: area too small (<=280).
Region 14: area = 304. Width/Height = 0.47. Accepted.
Region 15: area = 12. Rejected: area too small (<=280).
```




```
ROI at (813, 0, 843, 14) classified as 100.0. Min distance: 1.165. Accepted.
```

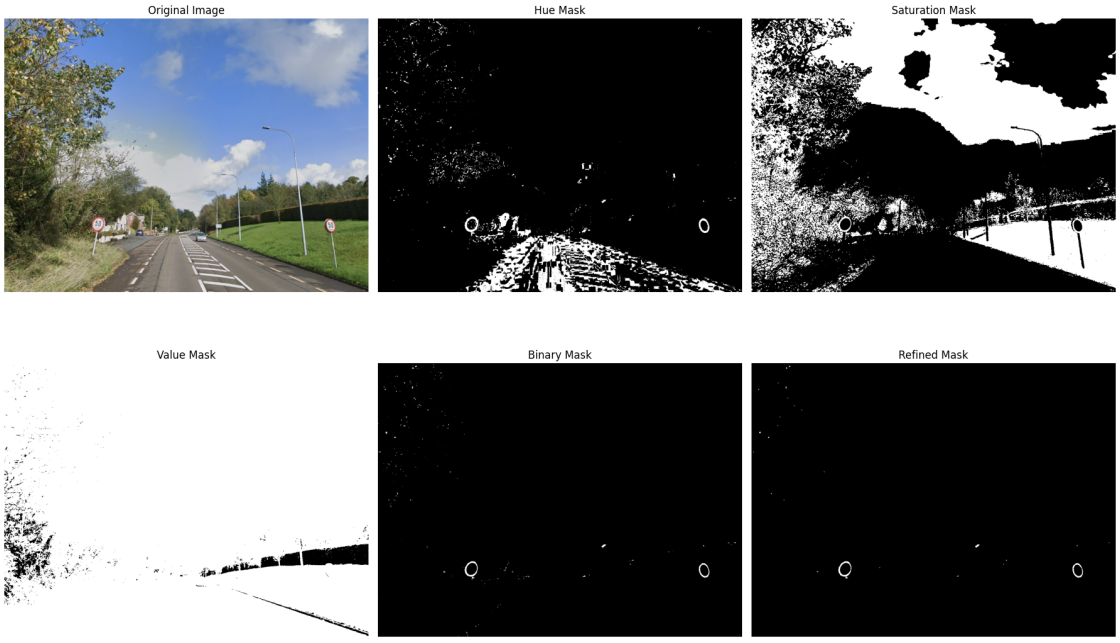
```
Detected 1 speed sign: 100.0 km/h
→ Now processing: image\50-0003x1.png
Region 0: area = 39. Rejected: area too small (<=280).
Region 1: area = 1896. Width/Height = 1.02. Accepted.
Region 2: area = 27. Rejected: area too small (<=280).
```




```
ROI at (372, 241, 471, 342) classified as 50.0. Min distance: 0.718. Accepted.
```


```
Detected 1 speed sign: 50.0 km/h
→ Now processing: image\50-0004x2.png
Region 0: area = 15. Rejected: area too small (<=280).
Region 1: area = 15. Rejected: area too small (<=280).
Region 2: area = 5. Rejected: area too small (<=280).
Region 3: area = 18. Rejected: area too small (<=280).
Region 4: area = 8. Rejected: area too small (<=280).
Region 5: area = 5. Rejected: area too small (<=280).
Region 6: area = 5. Rejected: area too small (<=280).
Region 7: area = 5. Rejected: area too small (<=280).
Region 8: area = 5. Rejected: area too small (<=280).
Region 9: area = 5. Rejected: area too small (<=280).
Region 10: area = 24. Rejected: area too small (<=280).
Region 11: area = 15. Rejected: area too small (<=280).
Region 12: area = 5. Rejected: area too small (<=280).
Region 13: area = 5. Rejected: area too small (<=280).
Region 14: area = 8. Rejected: area too small (<=280).
Region 15: area = 5. Rejected: area too small (<=280).
Region 16: area = 5. Rejected: area too small (<=280).
Region 17: area = 91. Rejected: area too small (<=280).
```

```
Region 18: area = 8. Rejected: area too small (<=280).
Region 19: area = 8. Rejected: area too small (<=280).
Region 20: area = 722. Width/Height = 0.75. Accepted.
Region 21: area = 524. Width/Height = 0.73. Accepted.
Region 22: area = 5. Rejected: area too small (<=280).
Region 23: area = 5. Rejected: area too small (<=280).
Region 24: area = 11. Rejected: area too small (<=280).
Region 25: area = 5. Rejected: area too small (<=280).
Region 26: area = 8. Rejected: area too small (<=280).
```




```
ROI at (695, 305, 756, 351) classified as 50.0. Min distance: 0.970. Accepted.
ROI at (702, 1128, 753, 1165) classified as 50.0. Min distance: 0.643. Accepted.
```


```
Detected 2 speed signs: 50.0 km/h, 50.0 km/h
→ Now processing: image\50-0005x1.png
Region 0: area = 5. Rejected: area too small (<=280).
Region 1: area = 21. Rejected: area too small (<=280).
Region 2: area = 5. Rejected: area too small (<=280).
Region 3: area = 5. Rejected: area too small (<=280).
Region 4: area = 5. Rejected: area too small (<=280).
Region 5: area = 37. Rejected: area too small (<=280).
Region 6: area = 15. Rejected: area too small (<=280).
Region 7: area = 56. Rejected: area too small (<=280).
Region 8: area = 8. Rejected: area too small (<=280).
Region 9: area = 8. Rejected: area too small (<=280).
Region 10: area = 10. Rejected: area too small (<=280).
Region 11: area = 625. Width/Height = 0.79. Accepted.
Region 12: area = 83. Rejected: area too small (<=280).
Region 13: area = 16. Rejected: area too small (<=280).
Region 14: area = 8. Rejected: area too small (<=280).
Region 15: area = 14. Rejected: area too small (<=280).
Region 16: area = 105. Rejected: area too small (<=280).
Region 17: area = 5. Rejected: area too small (<=280).
```

```
Region 18: area = 5. Rejected: area too small (<=280).
Region 19: area = 5. Rejected: area too small (<=280).
Region 20: area = 59. Rejected: area too small (<=280).
Region 21: area = 5. Rejected: area too small (<=280).
```