

**VIETNAM GENERAL CONFEDERATION OF LABOR  
TON DUC THANG UNIVERSITY  
FACULTY OF INFORMATION TECHNOLOGY**



**NGUYỄN THÀNH LONG – 522H0142  
PHẠM ĐẶNG THANH TRUNG – 522H0148**

## **FINAL TERM ESSAY**

# **INTRODUCTION TO DIGITAL IMAGE PROCESSING**

**HO CHI MINH CITY, 2024  
VIETNAM GENERAL CONFEDERATION OF LABOR  
TON DUC THANG UNIVERSITY  
FACULTY OF INFORMATION TECHNOLOGY**



**NGUYỄN THÀNH LONG – 522H0142**  
**PHẠM ĐẶNG THANH TRUNG – 522H0148**

## **FINAL TERM ESSAY**

# **INTRODUCTION TO DIGITAL IMAGE PROCESSING**

Advised by  
**Dr. Trinh Hung Cuong**

**HO CHI MINH CITY, 2024**

## ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Mr. Trinh Hung Cuong, our instructor and mentor, for his invaluable guidance and support throughout the final-term report of our project on Digital Image Processing. His expertise and patient approach in providing constructive feedback and suggestions have greatly improved our understanding of the subject. He has encouraged us to explore new technologies and techniques to enhance our image processing skills and project outcomes. We have learned a tremendous amount from his knowledge and experience in digital image processing and related fields. We are honored and privileged to have him as our teacher and supervisor.

*Ho Chi Minh city, 3<sup>rd</sup> December 2024.*

*Author*

*(Signature and full name)*

***Long***

Nguyen Thanh Long

***Trung***

Pham Dang Thanh Trung

## DECLARATION OF AUTHORSHIP

We hereby declare that this is our own project and is guided by Mr. Trinh Hung Cuong; The content research and results contained herein are central and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the main author from different sources, which are clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

**If something wrong happens, we'll take full responsibility for the content of my project.** Ton Duc Thang University is not related to the infringing rights, the copyrights that We give during the implementation process (if any).

*Ho Chi Minh city, 3<sup>rd</sup> December 2024*

*Author*

*(Signature and full name)*

***Long***

Nguyen Thanh Long

***Trung***

Pham Dang Thanh Trung

## TABLE OF CONTENT

### SUMMARY 1

1. Chapter 1: Methodology of Solving Tasks .....	1
2. Chapter 2: Task results.....	1
Chapter 1: Methodology of Solving Tasks.....	1
Chapter 2: Task results .....	26
1. Video 1:.....	26
2. Video 2:.....	30
REFERENCES .....	33

## SUMMARY

### 1. Chapter 1: Methodology of Solving Tasks

- Input: an input video (attached in the assignment), video1.mp4, video2.mp4.
- Processing: Draw rectangles surrounding each Traffic sign in all frames of the input videos automatically, then infer the content of the traffic sign automatically and output it above or below the sign. Finally, save the outputs into new video files.
- Summary solution:
  - Data preprocessing (cleaning data, converting formats, ...)
  - Algorithms implementation (using loops, functions, conditions)
  - Output generation (displaying results, saving files)
- Output: another version of the input video “video1.mp4” with rectangles surrounding each traffic sign and infer the content of the traffic sign automatically, “522H0142\_522H0148\_video1.avi”, “522H0142\_522H0148\_video2.avi”

### 2. Chapter 2: Task results

- Draw rectangles surrounding each Traffic sign in all frames of the input videos automatically, then infer the content of the traffic sign automatically and output it above or below the sign. Finally, save the outputs into new video files.

## Chapter 1: Methodology of Solving Tasks

- ❖ Template3 folder has 3 sub-folder:

circles	12/13/2024 9:56 AM	File folder
rectangles	12/13/2024 9:56 AM	File folder
triangles	12/13/2024 9:56 AM	File folder

Figure 1: template folder

- Sub-folder circles includes circle traffic sign:

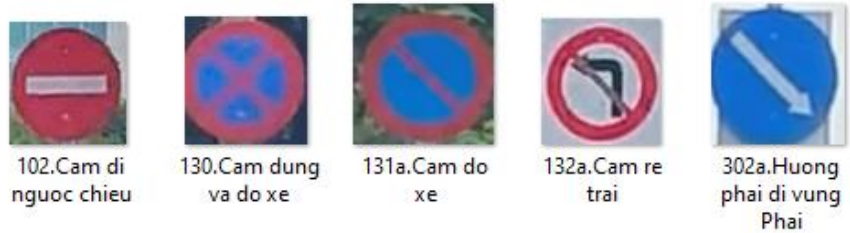


Figure 2: sub-folder-circles

- Sub-folder rectangles includes rectangle traffic sign:



Figure 3: sub-folder-rectangles









- Sub-folder triangles includes triangle traffic sign:



Figure 4: sub-folder-triangles

❖ Traffic signs in video 1 and video 2:

No	Name of traffic signs	Image of traffic signs
----	-----------------------	------------------------

1	Cấm đi ngược chiều	
2	Cấm dừng và đỗ xe	
3	Cấm đỗ xe	
4	Cấm rẽ trái	
5	Hướng phải đi vùng Phải	
6	Thẳng hoặc phải	
7	Hướng đi theo vạch kẻ đường	
8	Biển báo có nhiều chỗ ngoặt nguy hiểm vòng bên phải	



9	Cảnh báo trẻ em	
10	Đi chậm	

Figure 5: All traffic signs in video 1 and video 2.

❖ In the code, we specify the traffic sign code as follows:

```
...
Các mã biển báo giao thông:
102: Cấm đi ngược chiều
130: Cấm dừng và đỗ
131a: Cấm đỗ xe
132a: Cấm rẽ trái
302a: Hướng phải đi vùng phải
100: Thẳng hoặc phải (cấm ngược chiều)
411: Hướng đi theo vạch kẻ đường
202: Biển báo có nhiều chỗ ngoặt nguy hiểm bên phải
225: Biển báo cảnh báo có trẻ em
243: Biển báo cảnh báo hãy đi chậm

# Các mã biển báo trên được lấy theo chuẩn của bộ GTVT Việt Nam
```

❖ Code:

- Template matching methodology:

```
# Giới thiệu các phương pháp template matching
'''
Template Matching:
-----
|Phương pháp      | Giá trị tốt nhất | Đặc điểm |
|cv2.TM_CCOEFF    | Giá trị cao      | Phù hợp khi độ sáng/tối thay đổi. |
|cv2.TM_CCOEFF_NORMED | Giá trị gần 1 | Tương tự TM_CCOEFF, nhưng được chuẩn hóa. |
|cv2.TM_CCORR     | Giá trị cao      | Phù hợp cho các mẫu đơn giản. |
|cv2.TM_CCORR_NORMED | Giá trị gần 1 | Tương tự TM_CCORR, nhưng được chuẩn hóa. |
|cv2.TM_SQDIFF    | Giá trị thấp     | Tốt cho sự khác biệt nhỏ. |
|cv2.TM_SQDIFF_NORMED | Giá trị gần 0 | Tương tự TM_SQDIFF, nhưng được chuẩn hóa. |
-----
'''

Để xác định biến báo giao thông
=> Sử dụng các hàm dk nhận diện các đặc điểm riêng biệt của biến báo giao thông + kết hợp Template Matching để xác định mã biến báo

# Vì có các biến báo khác giống nhau đến >90% + Màu sắc clip ko rõ ràng nên việc chỉ dùng template matching là ko đủ chính xác
=> TM chủ yếu xác định cho biến tam giác và các biến hình chữ nhật
```

- Import library:

```
import cv2
import numpy as np
import os
```

Figure 6: Library

- For chapter 1, we use only 3 libraries: OpenCV, numpy and os.

- “clean\_images” function:

```
#Clean ảnh
def clean_images():
    file_list=os.listdir('./')
    for file_name in file_list:
        if '.png' in file_name:
            os.remove(file_name)
```

- **Purpose:** Deletes all .png image files in the current directory.
- **Usage:** Can be used to clean up temporary files before or after processing.

- “preprocess\_image” function:

```

#Hàm tiền xử lý ảnh (màu sắc) cho video đầu vào
def preprocess_image(image):

    # Chuyển ảnh sang hệ màu HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Màu đỏ (red color)
    lower_red1 = np.array([0, 150, 50])
    upper_red1 = np.array([10, 255, 150])
    lower_red2 = np.array([150, 100, 20])
    upper_red2 = np.array([180, 255, 150])
    mask_red1 = cv2.inRange(hsv, lower_red1, upper_red1)
    mask_red2 = cv2.inRange(hsv, lower_red2, upper_red2)
    mask_red = cv2.bitwise_or(mask_red1, mask_red2)

    # Màu xanh da trời (blue color)
    lower_blue = np.array([105, 100, 120])
    upper_blue = np.array([110, 255, 255])
    mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)

    # Màu vàng (yellow color) - Màu vàng cho biển báo giao thông nguy hiểm tam giác
    lower_yellow = np.array([20, 100, 100])
    upper_yellow = np.array([40, 255, 255])
    mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)

    # Làm mịn và xử lý mặt nạ
    mask_red = cv2.GaussianBlur(mask_red, (5, 5), 0)
    mask_red = cv2.erode(mask_red, None, iterations=2)
    mask_red = cv2.dilate(mask_red, None, iterations=2)

    mask_blue = cv2.GaussianBlur(mask_blue, (5, 5), 0)
    mask_blue = cv2.erode(mask_blue, None, iterations=2)
    mask_blue = cv2.dilate(mask_blue, None, iterations=2)

    mask_yellow = cv2.GaussianBlur(mask_yellow, (5, 5), 0)
    mask_yellow = cv2.erode(mask_yellow, None, iterations=2)
    mask_yellow = cv2.dilate(mask_yellow, None, iterations=2)

    return mask_red, mask_blue, mask_yellow

```

- **Purpose:** Preprocesses the input image to isolate red, blue, and yellow regions using HSV color space.
- **Details:**
  - **Color Conversion:** Converts the image from BGR to HSV for easier color segmentation.
  - **Color Masks:** Uses `cv2.inRange` to create masks for red, blue, and yellow colors.
  - **Mask Smoothing:** Applies Gaussian Blur, erosion, and dilation to clean up the masks, removing noise and unwanted details.
- “detect\_circle\_red” function:

```

# Hàm xử lý hình tròn màu đỏ
def detect_circle_red(contour):

    area=cv2.contourArea(contour)

    #Loại trừ các hình tròn nhỏ, tránh phát hiện sai
    if area<500:
        return False

    if area>1640 and area<2100:
        return False

    perimeter=cv2.arcLength(contour,True)

    if perimeter == 0:
        return False

    x,y,w,h=cv2.boundingRect(contour)
    aspect_ratio=float(w)/h
    circularity=(4*np.pi*area)/(perimeter**2)

    #Danh sách điều kiện
    dks=[
        {
            "circularity_range": (0.65,1),
            "aspect_ratio_range": (0.75,1.2),
            "height_range": (37,120),
            "perimeter_min": 175
        },
        {
            "circularity_range": (0.23,0.24),
            "aspect_ratio_range": (0.75,1.2),
            "height_range": (37,120),
            "area_range": (595,700),
            "perimeter_range": (165,180)
        },
        {
            "circularity_range": (0.12,0.13),
            "area_range": (1590,1658),
            "perimeter_range": (390,420),
        }
    ]

    #Kiểm tra
    for dk in dks:
        if (dk.get("circularity_range",(0,1))[0]<=circularity<=dk.get("circularity_range",(0,1))[1] and
            dk.get("aspect_ratio_range",(0,float("inf")))[0]<=aspect_ratio<=dk.get("aspect_ratio_range",(0,float("inf")))[1] and
            dk.get("height_range",(0,float("inf")))[0]<h<dk.get("height_range",(0,float("inf")))[1] and
            perimeter>dk.get("perimeter_min",0) and
            dk.get("area_range",(0,float("inf")))[0]<=area<=dk.get("area_range",(0,float("inf")))[1] and
            dk.get("perimeter_range",(0,float("inf")))[0]<=perimeter<=dk.get("perimeter_range",(0,float("inf")))[1]):
            return True

    return False

```

- **Purpose:** Detects red circular shapes based on geometric features like area, perimeter, aspect ratio, and circularity.
- **Details:**
  - **Area Filtering:** Excludes contours that are too small or within a specific unwanted area range to prevent false detections.
  - **Perimeter Calculation:** Calculates the perimeter of the contour and ensures it's not zero to avoid division errors.
  - **Bounding Box and Ratios:** Computes the bounding rectangle to determine aspect ratio and circularity.
  - **Condition Checking:** Iterates through predefined conditions to verify if the contour matches a valid red circle.

- “detect\_circle\_blue” function:

```
#Hàm xử lý hình tròn màu xanh
def detect_circle_blue(contour):
    area=cv2.contourArea(contour)
    if area<2300:
        return False

    perimeter=cv2.arcLength(contour,True)
    if perimeter == 0:
        return False

    x,y,w,h=cv2.boundingRect(contour)
    aspect_ratio=float(w)/h
    circularity=(4*np.pi*area)/(perimeter**2)

    #Điều kiện cho các loại hình tròn khác nhau
    small_circle=0.67<=circularity<=1 and 0.9<=aspect_ratio<=1.2 and 37<h<150
    medium_circle=0.36<=circularity<0.67 and 0.9<=aspect_ratio<=1.2 and 37<h<150 and area>8500 and perimeter>500
    large_circle=0.25<=circularity<0.36 and 0.9<=aspect_ratio<=1.2 and 37<h<150 and area>14500 and perimeter>700

    #Điều kiện loại trừ
    exclusion_area_perimeter=area<2500 and perimeter<210

    #Kiểm tra các điều kiện
    if exclusion_area_perimeter:
        return False
    if small_circle or medium_circle or large_circle:
        return True

    return False
```

- **Purpose:** Detects blue circular shapes based on geometric features similar to detect\_circle\_red.
- **Details:**
  - **Area and Perimeter Filtering:** Ensures the contour is large enough and has a valid perimeter.
  - **Bounding Box and Ratios:** Computes aspect ratio and circularity.
  - **Condition Checking:** Determines if the contour matches any predefined circle type (small, medium, large) and excludes unwanted cases.

- “detect\_triangle\_red” function:

```

#Hàm xử lí hình tam giác màu đỏ, có vẽ màu vàng hể tốt hơn, do video hơi kém chất lượng ánh sáng nền đỏ ko ra đỏ
def detect_triangle_red(contour):

    area=cv2.contourArea(contour)

    if area<2300:
        return False

    perimeter=cv2.arcLength(contour,True)
    approx=cv2.approxPolyDP(contour,0.04*perimeter,True)

    if len(approx) == 3: #Hình tam giác có 3 cạnh

        x,y,w,h=cv2.boundingRect(approx)
        aspect_ratio=float(w)/h

        #Điều kiện loại trừ cho hình tam giác nhỏ
        if area<1400 and perimeter<150:
            return False

        #Điều kiện cho hình tam giác hợp lệ
        valid_triangle=0.9<aspect_ratio<1 and 30<w<150 and 30<h<150

        if valid_triangle:
            return True

    return False

```

- **Purpose:** Detects red triangular shapes based on geometric features.
- **Details:**
  - **Area and Perimeter Filtering:** Excludes contours that are too small.
  - **Polygon Approximation:** Uses `cv2.approxPolyDP` to approximate the contour to a polygon and checks if it has 3 sides.
  - **Bounding Box and Ratios:** Computes aspect ratio and checks if it falls within the valid range for triangles.
  - **Condition Checking:** Ensures the triangle meets the size and shape criteria.
- “detect\_rectangle\_blue” function:

```

# Hàm xử lí hình chữ nhật màu xanh
def detect_rectangle_blue(contour):

    area=cv2.contourArea(contour)

    if area<1700:
        return False

    perimeter=cv2.arcLength(contour,True)
    approx=cv2.approxPolyDP(contour,0.02*perimeter,True)

    if len(approx)!=4:
        return False

    x,y,w,h=cv2.boundingRect(approx)
    aspect_ratio=float(w)/h

    #Cac dieu kien kích thước cho hình chu nhật mong muốn
    large_rectangle=w<150 and area>19000
    medium_rectangle=44<w<90 and 32<h<60 and 1200<area<6000 and 140<perimeter<400
    unwanted_rectangle=95<w<153 and 50<h<86 and perimeter<460 and area<8300

    #Loại trừ các trường hợp chu vi và diện tích không phù hợp
    high_perimeter_exclusion=perimeter>700 and area<10000
    low_area_exclusion=perimeter>100 and area<900

    #Cac dieu kien tỷ lệ khung hình và kích thước
    small_aspect_ratio=0.9<aspect_ratio<2 and 20<w<90 and 20<h<185
    large_aspect_ratio=0.9<aspect_ratio<2 and 90<w<300 and 20<h<185

    #Kiểm tra các điều kiện loại trừ và mong muốn
    if large_rectangle or medium_rectangle:
        return True

    if unwanted_rectangle or high_perimeter_exclusion or low_area_exclusion:
        return False

    if small_aspect_ratio or large_aspect_ratio:
        return True

    return False

```

- **Purpose:** Detects blue rectangular shapes based on geometric features.
- **Details:**
  - **Area and Perimeter Filtering:** Ensures the contour is large enough.
  - **Polygon Approximation:** Approximates the contour to a polygon and checks if it has 4 sides.
  - **Bounding Box and Ratios:** Computes aspect ratio and checks against predefined size conditions.
  - **Condition Checking:** Determines if the contour matches desired rectangle types and excludes unwanted cases.

- “detect\_triangle\_yellow” function:

```
# Hàm xử lí hình tam giác màu vàng
def detect_triangle_yellow(contour):
    # Tính diện tích của contour
    area = cv2.contourArea(contour)
    if area < 700: # Giảm ngưỡng loại bỏ các contour quá nhỏ
        return False

    if area > 10370 and area < 10380:
        return False

    # Tính chu vi và xác định các đỉnh
    perimeter = cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, 0.04 * perimeter, True)

    if len(approx) == 3: # Kiểm tra hình tam giác
        x, y, w, h = cv2.boundingRect(approx)
        aspect_ratio = float(w) / h

        # Kiểm tra tam giác hợp lệ dựa trên tỷ lệ và kích thước
        valid_triangle = (
            1 < aspect_ratio < 1.3 and
            20 < w < 300 and 20 < h < 300 and
            area > 700 and perimeter > 50
        )
        return valid_triangle

    return False
```

- **Purpose:** Detects yellow triangular shapes based on geometric features.
- **Details:**
  - **Area Filtering:** Excludes very small or specific unwanted contour areas.
  - **Polygon Approximation:** Uses cv2.approxPolyDP to check for 3-sided polygons.
  - **Bounding Box and Ratios:** Computes aspect ratio and ensures it falls within the valid range.
  - **Condition Checking:** Validates the triangle based on size and shape criteria.



- “detect\_traffic\_signs” function:

```
#Hàm tìm biển báo giao thông
def detect_traffic_signs(image):
    mask_red, mask_blue, mask_yellow = preprocess_image(image)
    contours_red, _ = cv2.findContours(mask_red, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours_blue, _ = cv2.findContours(mask_blue, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours_yellow, _ = cv2.findContours(mask_yellow, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    traffic_signs = []
    red_bboxes = []

    # Check red contours (circle and triangle)
    for contour in contours_red:
        if detect_circle_red(contour):
            x, y, w, h = cv2.boundingRect(contour)
            area = cv2.contourArea(contour)
            perimeter = cv2.arcLength(contour, True)
            circularity = (4 * np.pi * area) / (perimeter**2)
            red_bboxes.append((x, y, w, h))
            traffic_signs.append(('Red Circle', x, y, w, h, f'Area: {area} px, Perimeter: {perimeter} px, Circularity: {circularity:.2f}'))

        elif detect_triangle_red(contour):
            x, y, w, h = cv2.boundingRect(contour)
            area = cv2.contourArea(contour)
            perimeter = cv2.arcLength(contour, True)
            red_bboxes.append((x, y, w, h))
            traffic_signs.append(('Red Triangle', x, y, w, h, f'Area: {area} px, Perimeter: {perimeter} px'))

    # Check blue contours (circle, rectangle, square)
    for contour in contours_blue:
        x, y, w, h = cv2.boundingRect(contour)
        is_inside_red_area = any(rx <= x <= rx + rw and ry <= y <= ry + rh for rx, ry, rw, rh in red_bboxes)

        if not is_inside_red_area:
            area = cv2.contourArea(contour)
            perimeter = cv2.arcLength(contour, True)
            circularity = (4 * np.pi * area) / (perimeter**2)

            if detect_circle_blue(contour):
                # Check if the circle is pure blue and does not have any red in the region
                cropped_image = image[y:y+h, x:x+w]
                mask_red_cropped = cv2.inRange(cropped_image, (0, 0, 100), (50, 50, 255)) # Range for red color
                red_area = cv2.countNonZero(mask_red_cropped)

                if red_area == 0: # No red in the area, this is a pure blue circle
                    traffic_signs.append(('Blue Circle', x, y, w, h, f'Area: {area} px, Perimeter: {perimeter} px, Circularity: {circularity:.2f}'))

            elif detect_rectangle_blue(contour):
                traffic_signs.append(('Blue Rectangle', x, y, w, h, f'Width: {w} px, Height: {h} px, Area: {area} px, Perimeter: {perimeter} px'))

    # Xử lý biển báo màu vàng
    for contour in contours_yellow:
        x, y, w, h = cv2.boundingRect(contour)
        is_inside_red_area = any(rx <= x <= rx + rw and ry <= y <= ry + rh for rx, ry, rw, rh in red_bboxes)

        # Chỉ xử lý nếu contour vàng nằm trong viền đỏ
        if detect_triangle_yellow(contour):
            area = cv2.contourArea(contour)
            perimeter = cv2.arcLength(contour, True)
            traffic_signs.append(('Red-Yellow Triangle', x-20, y-20, w+50, h+50, f'Area: {area} px, Perimeter: {perimeter} px'))

    return traffic_signs
```

- **Purpose:** Detects and lists traffic signs within an image frame.
- **Details:**
  - **Preprocessing:** Uses preprocess\_image to create color masks.
  - **Contour Detection:** Uses cv2.findContours to find contours in red, blue, and yellow masks.
  - **Red Sign Detection:** Checks each red contour to determine if it's a red circle or red triangle, storing bounding boxes and sign information.
  - **Blue Sign Detection:** Checks each blue contour, excluding those within red areas, to determine if it's a blue circle or blue rectangle, and appends to the sign list.

- **Yellow Sign Detection:** Checks yellow contours within red boundaries to identify red-yellow triangles and appends to the sign list.

- “Detect\_sign130” function:

```
# Hàm xác định biển báo 130
def Detect_sign130(image):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Định nghĩa khoảng màu đỏ
    mask_red1 = cv2.inRange(hsv, (0, 50, 50), (10, 255, 255))
    mask_red2 = cv2.inRange(hsv, (160, 50, 50), (180, 255, 255))
    mask_red = cv2.bitwise_or(mask_red1, mask_red2)

    # Tìm vùng đỏ lớn nhất (vùng biển)
    contours_red, _ = cv2.findContours(mask_red, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if len(contours_red) == 0:
        return False
    largest_red_contour = max(contours_red, key=cv2.contourArea)
    red_area = cv2.contourArea(largest_red_contour)
    if red_area == 0:
        return False

    # Tạo mask vùng đỏ chính
    mask_red_main = np.zeros_like(mask_red)
    cv2.drawContours(mask_red_main, [largest_red_contour], -1, 255, -1)

    # Định nghĩa khoảng màu xanh dương
    lower_blue = np.array([100, 50, 50])
    upper_blue = np.array([130, 255, 255])
    mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)

    # Lấy phần xanh nằm bên trong vùng đỏ
    mask_blue_inside = cv2.bitwise_and(mask_blue, mask_red_main)

    # Tìm các vùng xanh bên trong vùng đỏ
    contours_blue, _ = cv2.findContours(mask_blue_inside, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Ngưỡng diện tích tối thiểu cho vùng xanh (5% diện tích vùng đỏ)
    min_ratio = 0.05
    min_area = min_ratio * red_area

    # Đếm số vùng xanh đủ lớn
    valid_blue_count = 0
    for cnt in contours_blue:
        blue_area = cv2.contourArea(cnt)
        if blue_area >= min_area:
            valid_blue_count += 1

    # Kiểm tra số vùng xanh thỏa mãn
    # Giả sử bạn muốn có 4 vùng xanh lớn hơn 5% vùng đỏ
    if valid_blue_count >= 3:
        return True
    else:
        return False
```

- **Purpose:** Determines if the image contains sign 130
- **Details:**
  - **Red Mask Creation:** Identifies red regions in the image.
  - **Largest Red Contour:** Assumes the largest red contour is the main sign area.

- **Blue Mask Within Red Area:** Identifies blue regions within the main red area.
- **Blue Region Counting:** Counts blue regions that are sufficiently large (at least 5% of the red area).
- **Decision:** Returns True if there are at least 3 large blue regions within the red area, indicating sign 130.

- “Detect\_sign131a” function:

```
# Hàm xác định biển báo 131a
def Detect_sign131a(image):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Định nghĩa khoảng màu đỏ
    mask_red1 = cv2.inRange(hsv, (0, 50, 50), (10, 255, 255))
    mask_red2 = cv2.inRange(hsv, (160, 50, 50), (180, 255, 255))
    mask_red = cv2.bitwise_or(mask_red1, mask_red2)

    # Tìm vùng đỏ lớn nhất (vùng biển)
    contours_red, _ = cv2.findContours(mask_red, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if len(contours_red) == 0:
        return False
    largest_red_contour = max(contours_red, key=cv2.contourArea)
    red_area = cv2.contourArea(largest_red_contour)
    if red_area == 0:
        return False

    # Tạo mask vùng đỏ chính
    mask_red_main = np.zeros_like(mask_red)
    cv2.drawContours(mask_red_main, [largest_red_contour], -1, 255, -1)

    # Định nghĩa khoảng màu xanh dương
    lower_blue = np.array([100, 50, 50])
    upper_blue = np.array([130, 255, 255])
    mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)

    # Lấy phần xanh nằm bên trong vùng đỏ
    mask_blue_inside = cv2.bitwise_and(mask_blue, mask_red_main)

    # Tìm các vùng xanh bên trong vùng đỏ
    contours_blue, _ = cv2.findContours(mask_blue_inside, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Ngưỡng diện tích tối thiểu cho vùng xanh (5% diện tích vùng đỏ)
    min_ratio = 0.05
    min_area = min_ratio * red_area

    # Đếm số vùng xanh đủ lớn
    valid_blue_count = 0
    for cnt in contours_blue:
        blue_area = cv2.contourArea(cnt)
        if blue_area >= min_area:
            valid_blue_count += 1

    # Kiểm tra số vùng xanh thỏa mãn
    # Giả sử bạn muốn có 4 vùng xanh lớn hơn 5% vùng đỏ
    if valid_blue_count >= 1 and valid_blue_count <= 3 :
        return True
    else:
        return False
```

- **Purpose:** Identifies sign 131a

- **Details:**

- **Red Mask Creation:** Similar to `Detect_sign130`, it creates a mask to identify red regions.
- **Largest Red Contour:** Assumes the largest red contour represents the main sign area.
- **Blue Mask Within Red Area:** Identifies blue regions inside the red sign area.
- **Blue Region Counting:** Counts blue regions that are at least 5% of the red area.
- **Decision:** Returns `True` if there are between 1 and 3 large blue regions within the red area, indicating sign 131a.

- “`detect_sign102`” function:

```
# hàm xác định biển báo 102
def detect_sign102(image):
    # Chuyển sang HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Khoảng màu đỏ
    mask_red1 = cv2.inRange(hsv, (0, 50, 50), (10, 255, 255))
    mask_red2 = cv2.inRange(hsv, (160, 50, 50), (180, 255, 255))
    mask_red = cv2.bitwise_or(mask_red1, mask_red2)

    # Khoảng màu xanh dương
    mask_blue = cv2.inRange(hsv, (100, 50, 50), (130, 255, 255))

    # Tìm contour lớn nhất trong mask_red (giả định đây là vùng biển đỏ chính)
    contours, _ = cv2.findContours(mask_red, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) == 0:
        # Không tìm thấy vùng đỏ nào
        return False

    # Chọn contour lớn nhất (diện tích lớn nhất)
    largest_contour = max(contours, key=cv2.contourArea)

    # Tạo mask trắng (0) rồi vẽ contour này lên để tạo mask vùng đỏ chính
    red_mask_shape = mask_red.shape
    mask_red_main = np.zeros(red_mask_shape, dtype=np.uint8)
    cv2.drawContours(mask_red_main, [largest_contour], -1, 255, -1) # vẽ contour filled

    # Lấy phần xanh nằm bên trong vùng đỏ
    mask_blue_inside = cv2.bitwise_and(mask_blue, mask_red_main)

    total_pixels = image.shape[0] * image.shape[1]
    red_pixels = cv2.countNonZero(mask_red)
    blue_inside_pixels = cv2.countNonZero(mask_blue_inside)

    red_ratio = red_pixels / total_pixels
    blue_inside_ratio = blue_inside_pixels / total_pixels

    # Kiểm tra điều kiện:
    # - blue_inside_ratio < 0.3 (tùy chỉnh ngưỡng)
    # - red_ratio > 0.3 (tùy chỉnh ngưỡng)
    if blue_inside_ratio < 0.4 and red_ratio > 0.6:
        return True
    else:
        return False
```

- **Purpose:** Identifies sign 102
  - **Details:**
    - **Red Mask Creation:** Identifies red regions in the image.
    - **Largest Red Contour:** Assumes the largest red contour is the main sign area.
    - **Blue Mask Within Red Area:** Identifies blue regions within the main red area.
    - **Pixel Ratio Calculation:** Calculates the ratio of red pixels to total pixels and blue pixels within red area to total pixels.
    - **Decision:** Returns True if the red area occupies more than 60% of the image and blue within red is less than 40%, indicating sign 102.
- 
- “detect\_sign\_302a” function:

```

# detect 302a - xét tròn
def detect_sign_302a(image):
    # Chuyển sang HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Mặt nạ màu đỏ
    mask_red1 = cv2.inRange(hsv, (0, 50, 50), (10, 255, 255))
    mask_red2 = cv2.inRange(hsv, (160, 50, 50), (180, 255, 255))
    mask_red = cv2.bitwise_or(mask_red1, mask_red2)

    # Mặt nạ màu xanh dương
    mask_blue = cv2.inRange(hsv, (100, 50, 50), (130, 255, 255))

    # Tính tỉ lệ màu
    total_pixels = image.shape[0] * image.shape[1]
    red_pixels = cv2.countNonZero(mask_red)
    blue_pixels = cv2.countNonZero(mask_blue)

    red_ratio = red_pixels / total_pixels
    blue_ratio = blue_pixels / total_pixels

    # Kiểm tra màu sắc trước
    if not (red_ratio < 0.05 and blue_ratio > 0.8):
        return False

    # Tìm contour lớn nhất trong mask xanh, giả định đó là biển chính
    contours, _ = cv2.findContours(mask_blue, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) == 0:
        return False
    largest_contour = max(contours, key=cv2.contourArea)

    # Tìm vòng tròn bao nhỏ nhất
    (x, y), radius = cv2.minEnclosingCircle(largest_contour)
    circle_area = np.pi * (radius ** 2)
    contour_area = cv2.contourArea(largest_contour)

    # Tỉ lệ diện tích contour so với diện tích vòng tròn bao quanh
    # Nếu tỉ lệ này gần 1, nghĩa là hình dạng gần tròn
    shape_ratio = contour_area / circle_area

    # Ngưỡng tùy chọn, ví dụ 0.8 nghĩa là contour chiếm ít nhất 80% diện tích vòng tròn
    if shape_ratio > 0.8:
        return True
    else:
        return False

```

- **Purpose:** Identifies sign 302a.
- **Details:**
  - **Color Mask Creation:** Identifies red and blue regions in the image.
  - **Color Ratio Calculation:** Calculates the ratio of red pixels to total pixels and blue pixels to total pixels.
  - **Preliminary Check:** Ensures that red is minimal (<5%) and blue dominates (>80%).
  - **Blue Contour Analysis:**
    - **Largest Blue Contour:** Assumes the largest blue contour represents the main sign area.

- **Enclosing Circle:** Finds the smallest circle that can enclose the contour.
- **Shape Ratio:** Compares the area of the contour to the area of the enclosing circle.
- **Decision:** Returns True if the contour area is at least 80% of the enclosing circle area, indicating a nearly circular blue sign, which corresponds to sign 302a.

- “detect\_sign123a” function:

```
# Hàm xác định biển cấm rẽ trái - 123a
def detect_sign123a(image):
    # Chuyển sang HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Tạo mask màu đen
    # Màu đen thường là vùng có độ sáng (V) thấp.
    # Ta có thể thử ngưỡng: V < 50 để nhận diện vùng đen
    # Đặt H,S từ (0,0) đến (180,255) không quan trọng lắm vì đen chủ yếu do V thấp
    mask_black = cv2.inRange(hsv, (0, 0, 0), (180, 255, 50))

    # Tạo mask màu xanh dương
    mask_blue = cv2.inRange(hsv, (100, 50, 50), (130, 255, 255))

    total_pixels = image.shape[0] * image.shape[1]
    black_pixels = cv2.countNonZero(mask_black)
    blue_pixels = cv2.countNonZero(mask_blue)

    black_ratio = black_pixels / total_pixels
    blue_ratio = blue_pixels / total_pixels

    # Điều kiện:
    # - black_ratio > 0.001
    # - blue_ratio gần như không có (ví dụ < 1%)
    if black_ratio > 0.001 and blue_ratio < 0.05:
        return True
    else:
        return False
```

- **Purpose:** Identifies sign 123a.
- **Details:**
  - **Black Mask Creation:** Detects black regions in the image by thresholding the V channel in HSV.
  - **Blue Mask Creation:** Identifies blue regions in the image.
  - **Pixel Ratio Calculation:** Calculates the ratio of black pixels and blue pixels to total pixels.

- **Decision:** Returns True if the image contains significant black regions ( $>0.1\%$ ) and minimal blue regions ( $<5\%$ ), indicating sign 123a.

- “has\_blue\_less\_than\_10\_percent” function:

```
# Hàm xác màu xanh dương chiếm < 10% diện tích
def has_blue_less_than_10_percent(image):
    # Chuyển ảnh sang không gian màu HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Định nghĩa khoảng màu xanh dương
    lower_blue = np.array([100, 50, 50])
    upper_blue = np.array([130, 255, 255])

    # Tạo mask để lọc màu xanh
    mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)

    # Tính số pixel trong ảnh và số pixel màu xanh
    total_pixels = image.shape[0] * image.shape[1]
    blue_pixels = cv2.countNonZero(mask_blue)

    # Tính tỷ lệ màu xanh
    blue_ratio = blue_pixels / total_pixels

    # Kiểm tra xem có bé hơn 10% hay không
    if blue_ratio < 0.1:
        return True
    else:
        return False
```

- **Purpose:** Checks if the blue color occupies less than 10% of the image area.
- **Details:**
  - **Blue Mask Creation:** Identifies blue regions in the image.
  - **Pixel Ratio Calculation:** Calculates the ratio of blue pixels to total pixels.
  - **Decision:** Returns True if blue occupies less than 10% of the image, otherwise False.



- “load\_templates” function:

```
#- - - - -
# XỬ LÝ NỘI DUNG BIẾN BẢO BẮNG TEMPLATE MATCHING
# Lấy các template từ thư mục templates
def load_templates(templates_dir, shape_type=None):
    templates = {}
    shape_dir = templates_dir

    if shape_type:
        shape_dir = os.path.join(templates_dir, shape_type)

    # Check if the shape directory exists
    if not os.path.exists(shape_dir):
        print(f"Cảnh báo: Thư mục hình dạng '{shape_dir}' không tìm thấy.")
        return templates

    # Load templates from the specified shape directory
    for template_name in os.listdir(shape_dir):
        template_path = os.path.join(shape_dir, template_name)
        template = cv2.imread(template_path)
        if template is not None:
            templates[template_name] = template

    return templates
```

- **Purpose:** Loads template images from a specified directory for use in Template Matching.
- **Details:**
  - **Shape Directory:** Optionally specifies a subdirectory based on shape type (circles, rectangles, triangles).
  - **Existence Check:** Prints a warning if the specified directory does not exist.
  - **Template Loading:** Reads each image file in the directory and stores it in a dictionary with the filename as the key.
- “match\_template” function:

```

# Tìm template phù hợp với biển báo giao thông, trả về name của template -> chính là tên biển báo cần tìm
def match_template(image, x, y, w, h, templates, sign_type):
    # Kiểm tra ảnh và vùng cắt hợp lệ
    if image is None or image.size == 0:
        return None
    if y + h > image.shape[0] or x + w > image.shape[1] or y < 0 or x < 0:
        return None

    # Cắt vùng ảnh chứa biển báo
    cropped_image = image[y:y + h, x:x + w]
    cropped_gray = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2GRAY)
    cropped_hsv = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2HSV)

    max_match_score = 0
    best_match_name = None

    for template_name, template in templates.items():
        template_hsv = cv2.cvtColor(template, cv2.COLOR_BGR2HSV)
        template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

        # Kiểm tra biển xanh tròn (302a) trên template
        template_is_blue_circle = detect_sign_302a(template)

        # Nếu sign_type là Blue Circle
        if "Blue Circle" in sign_type:
            # Kiểm tra red_pixels trên template
            mask_red1 = cv2.inRange(template_hsv, (0, 50, 50), (10, 255, 255))
            mask_red2 = cv2.inRange(template_hsv, (160, 50, 50), (180, 255, 255))
            red_pixels = cv2.countNonZero(mask_red1) + cv2.countNonZero(mask_red2)

            # Nếu template không đỏ và là blue circle hoặc không đỏ mà sign là Blue Circle
            if red_pixels == 0 or template_is_blue_circle:
                return os.path.splitext(template_name)[0]

            # Không khớp thì bỏ qua template này
            continue

        # Nếu template là Blue Circle, mà sign_type không phải Blue Circle, bỏ qua
        if template_is_blue_circle:
            continue

```

```

# Nếu sign_type là Red Circle
if "Red Circle" in sign_type:
    # Kiểm tra từng loại biển Red Circle:
    # 1. Biển 102
    if detect_sign102(template):
        if detect_sign102(cropped_image):
            return os.path.splitext(template_name)[0]
        else:
            continue

    # 2. Biển 130
    if Detect_sign130(template):
        if Detect_sign130(cropped_image):
            return os.path.splitext(template_name)[0]
        else:
            continue

    # 3. Biển 123a
    if detect_sign123a(template):
        if has_blue_less_than_10_percent(cropped_image):
            return os.path.splitext(template_name)[0]
        else:
            continue

    # 4. Còn lại là 131a
    if Detect_sign131a(template):
        return os.path.splitext(template_name)[0]

# Nếu không khớp gì, tiếp tục xét template khác
return None

# Nếu đến đây tức là sign_type không phải Blue Circle, không phải Red Circle
# Thực hiện template matching cơ bản:
if template_gray.shape != (h, w):
    resized_template = cv2.resize(template_gray, (w, h))
else:
    resized_template = template_gray

# Thực hiện template matching
match_result = cv2.matchTemplate(cropped_gray, resized_template, cv2.TM_CCOEFF)
_, max_val, _, _ = cv2.minMaxLoc(match_result)

# Cập nhật kết quả
if max_val > max_match_score:
    max_match_score = max_val
    best_match_name = template_name

# Dừng sớm nếu match quá tốt
if max_val > 0.95:
    break

# Trả về tên template nếu có
if best_match_name:
    best_match_name = os.path.splitext(best_match_name)[0]
return best_match_name

```

- **Purpose:** Compares the cropped region of the image containing the traffic sign with the loaded templates to identify the specific sign code.
- **Details:**

- **Validation:** Ensures the image and the cropped region are valid and within bounds.
  - **Cropping and Conversion:** Extracts the region of interest and converts it to grayscale and HSV.
  - **Template Comparison:**
    - **Blue Circle (302a):** Checks if the template is a blue circle and matches accordingly.
    - **Red Circles:** Specifically checks for sign types 102, 130, 123a, and 131a using their respective detection functions.
    - **Basic Template Matching:** For other sign types, performs standard template matching using `cv2.matchTemplate` and tracks the best match based on the highest correlation value.
  - **Result:** Returns the name of the best-matching template or None if no suitable match is found.
- 
- “draw\_and\_save\_traffic\_signs” function:

```

#- - - - -
# Bước cuối cùng: Vẽ và lưu video kết quả
def draw_and_save_traffic_signs(input_video_path, templates_dir='templates', output_video_path='output.avi', delay=10):
    # Mở video đầu vào
    cap = cv2.VideoCapture(input_video_path)

    if not cap.isOpened():
        print("Lỗi không thể mở video input.")
        return

    # Lấy các thuộc tính của video để lưu video đầu ra
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frame_rate = int(cap.get(cv2.CAP_PROP_FPS))

    # Định nghĩa codec và tạo đối tượng VideoWriter
    fourcc = cv2.VideoWriter_fourcc(*'MJPG') # Codec MJPG cho macOS
    out = cv2.VideoWriter(output_video_path, fourcc, frame_rate, (frame_width, frame_height))

    while True:
        ret, frame = cap.read()
        if not ret:
            # Kết thúc video nếu không còn frame
            break

        # Phát hiện biển báo giao thông
        traffic_signs = detect_traffic_signs(frame)

        # Lặp qua từng biển báo giao thông được phát hiện
        for sign_type, x, y, w, h, label in traffic_signs:
            # Xác định loại hình biển báo (giúp tối ưu hóa)
            shape_type = None
            if 'Circle' in sign_type:
                shape_type = 'circles'
            elif 'Rectangle' in sign_type or 'Square' in sign_type:
                shape_type = 'rectangles'
            elif 'Triangle' in sign_type:
                shape_type = 'triangles'

            # Tải các mẫu dựa trên loại hình
            templates = load_templates(templates_dir, shape_type)

            # Khớp biển báo với một mẫu
            matched_template = match_template(frame, x, y, w, h, templates, sign_type)

            # Vẽ khung bao quanh biển báo giao thông
            color = (0, 255, 0) if 'Blue' in sign_type else (0, 0, 255)
            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

            # Hiển thị nhãn của biển báo giao thông nếu có
            if matched_template:
                label_text = matched_template
                text_color = (255, 255, 255)
                text_size = cv2.getTextSize(label_text, cv2.FONT_HERSHEY_SIMPLEX, 0.9, 2)[0]
                text_x = x - text_size[0] if x - text_size[0] > 0 else 0
                cv2.putText(frame, label_text, (text_x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, text_color, 2, cv2.LINE_AA)

        # Hiển thị văn bản "522H0142_522H0148" ở góc dưới video, màu đỏ
        label_text = "522H0142_522H0148"
        text_color = (0, 0, 255) # Màu đỏ
        cv2.putText(frame, label_text, (frame_width - 350, frame_height - 20), cv2.FONT_HERSHEY_SIMPLEX, 1, text_color, 2, cv2.LINE_AA)

        # Lưu frame vào video đầu ra
        out.write(frame)

    # Nhấn 'q' để thoát nếu sử dụng imshow, nhưng chúng ta không cần imshow nữa

    # Giải phóng tài nguyên
    cap.release()
    out.release()
    cv2.destroyAllWindows()

```

- **Purpose:** Processes the input video to detect and recognize traffic signs, draws bounding boxes and labels on detected signs, and saves the processed video.
- **Details:**

- **Video Capture:** Opens the input video using `cv2.VideoCapture`.
- **Video Properties:** Retrieves frame width, height, and frame rate to set up the output video.
- **Video Writer:** Initializes `cv2.VideoWriter` to write the output video with the specified codec and properties.
- **Frame Processing Loop:**
  - **Frame Reading:** Reads each frame from the input video.
  - **Traffic Sign Detection:** Uses `detect_traffic_signs` to identify signs within the frame.
  - **Sign Processing:**
    - ✓ **Shape Determination:** Identifies the shape type (circle, rectangle, triangle) to load appropriate templates.
    - ✓ **Template Matching:** Matches detected signs with templates to identify the specific sign code.
    - ✓ **Drawing Bounding Boxes:** Draws rectangles around detected signs with green for blue signs and red for others.
    - ✓ **Labeling:** If a match is found, labels the sign with the template name.
  - **Fixed Label:** Adds the text "522H0142\_522H0148" at the bottom corner of the video in red.
  - **Frame Writing:** Writes the processed frame to the output video.
- **Resource Cleanup:** Releases video capture and writer objects and closes any OpenCV windows.

- “main”:

```

# Các tham số đầu vào
input_video_path_1 = 'video1.mp4'
input_video_path_2 = 'video2.mp4'
templates_dir = 'templates3'
output_video_path_1 = '522H0142_522H0148_video1.avi'
output_video_path_2 = '522H0142_522H0148_video2.avi'

# Xử lý video 1
draw_and_save_traffic_signs(input_video_path_1, templates_dir, output_video_path_1)

# Xử lý video 2
draw_and_save_traffic_signs(input_video_path_2, templates_dir, output_video_path_2)
#- - - - -

```

- **Purpose:** Defines the input and output video paths and initiates the processing of two videos.
- **Details:**
  - **Input Videos:** Specifies video1.mp4 and video2.mp4 as the input videos.
  - **Templates Directory:** Specifies templates3 as the directory containing template images.
  - **Output Videos:** Defines the output video filenames as 522H0142\_522H0148\_video1.avi and 522H0142\_522H0148\_video2.avi.
  - **Function Calls:** Calls draw\_and\_save\_traffic\_signs for each input video to perform detection, recognition, and saving of the processed video.

## Chapter 2: Task results

### 1. Video 1:

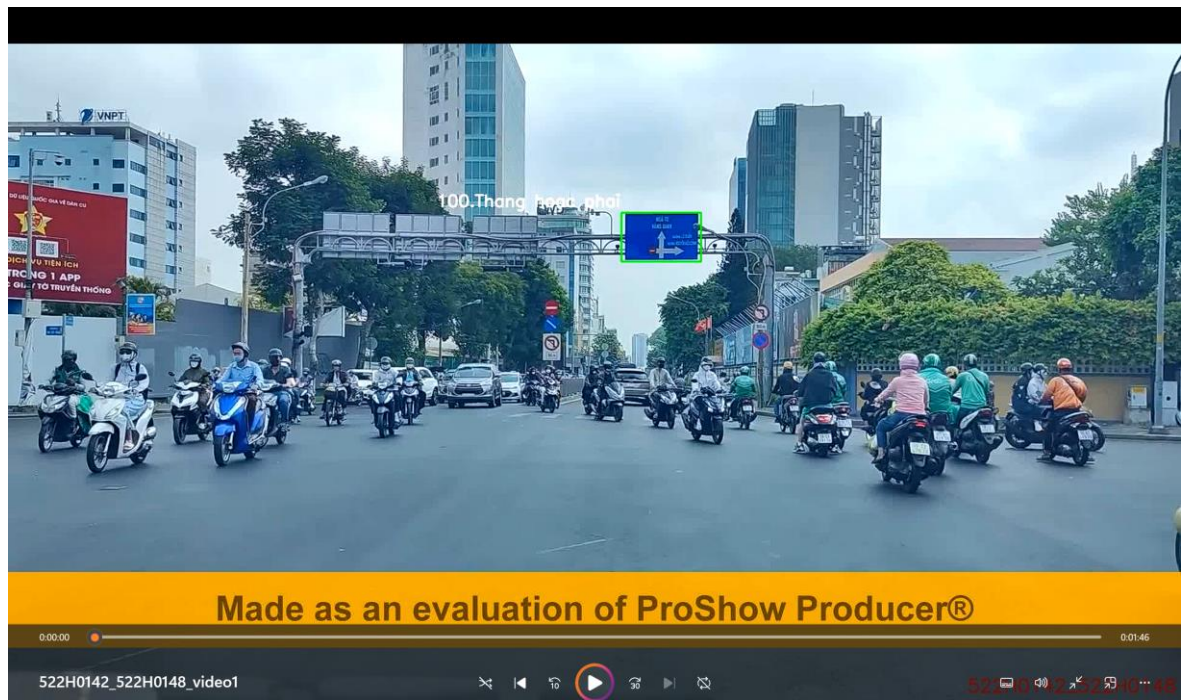


Figure 1.1: Image of video 1

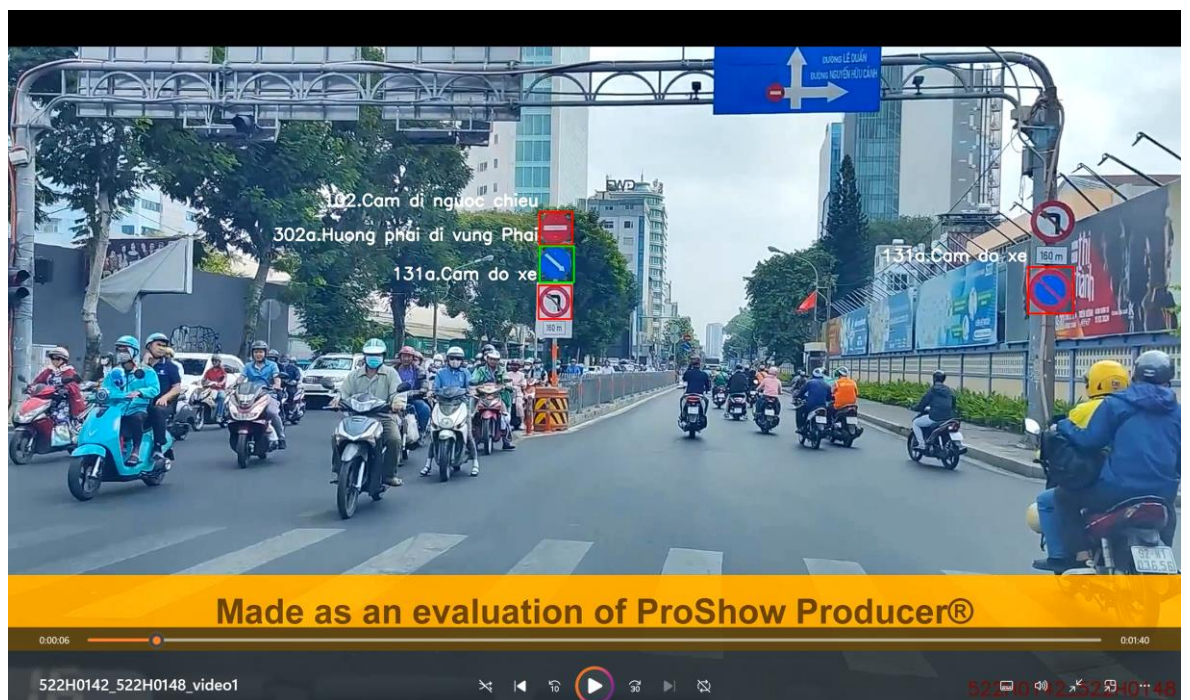


Figure 1.2: Image of video 1



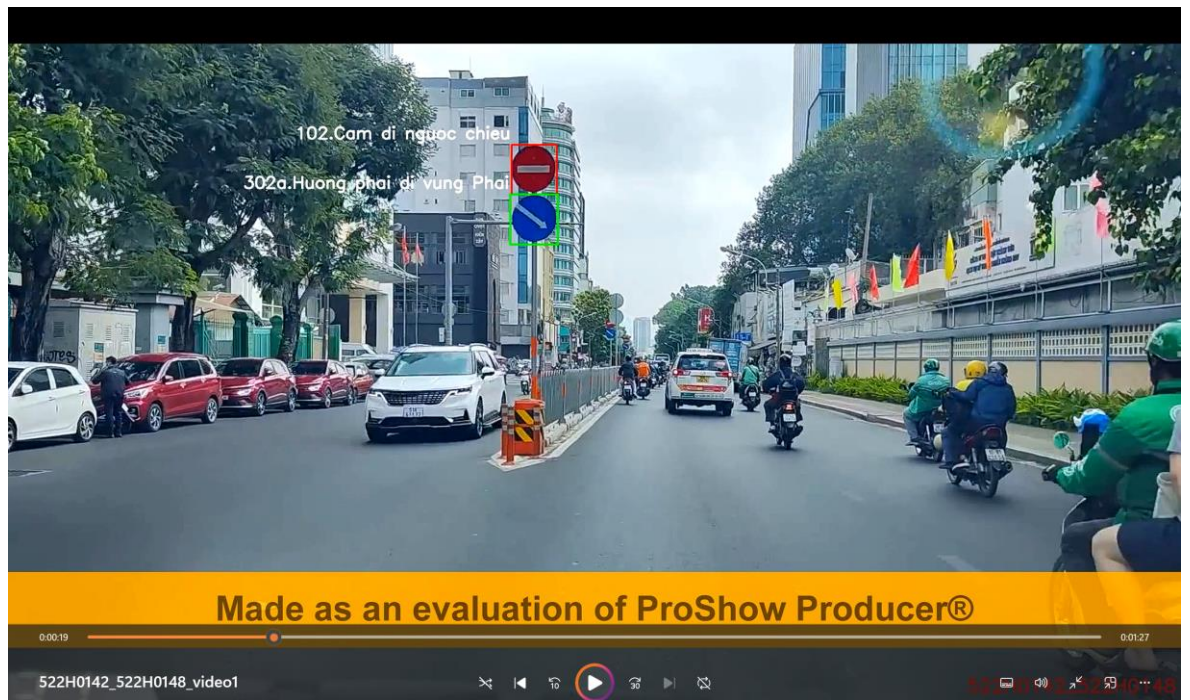


Figure 1.3: Image of video 1



Figure 1.4: Image of video 1



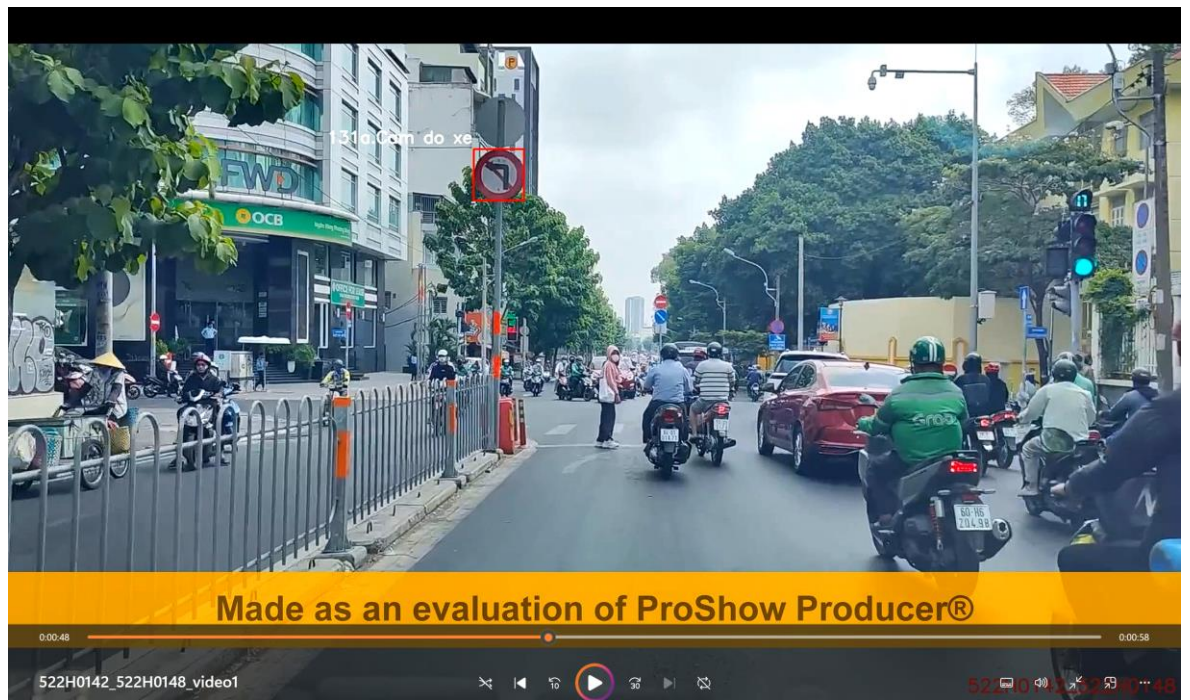


Figure 1.5: Image of video 1

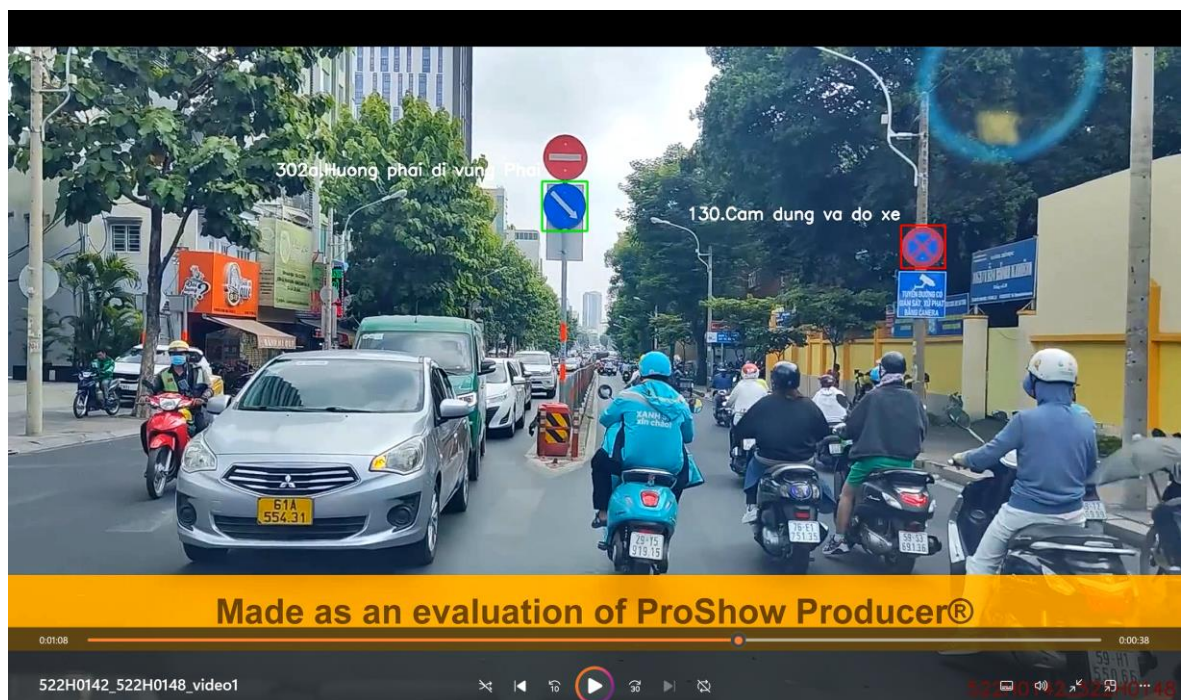


Figure 1.6: Image of video 1



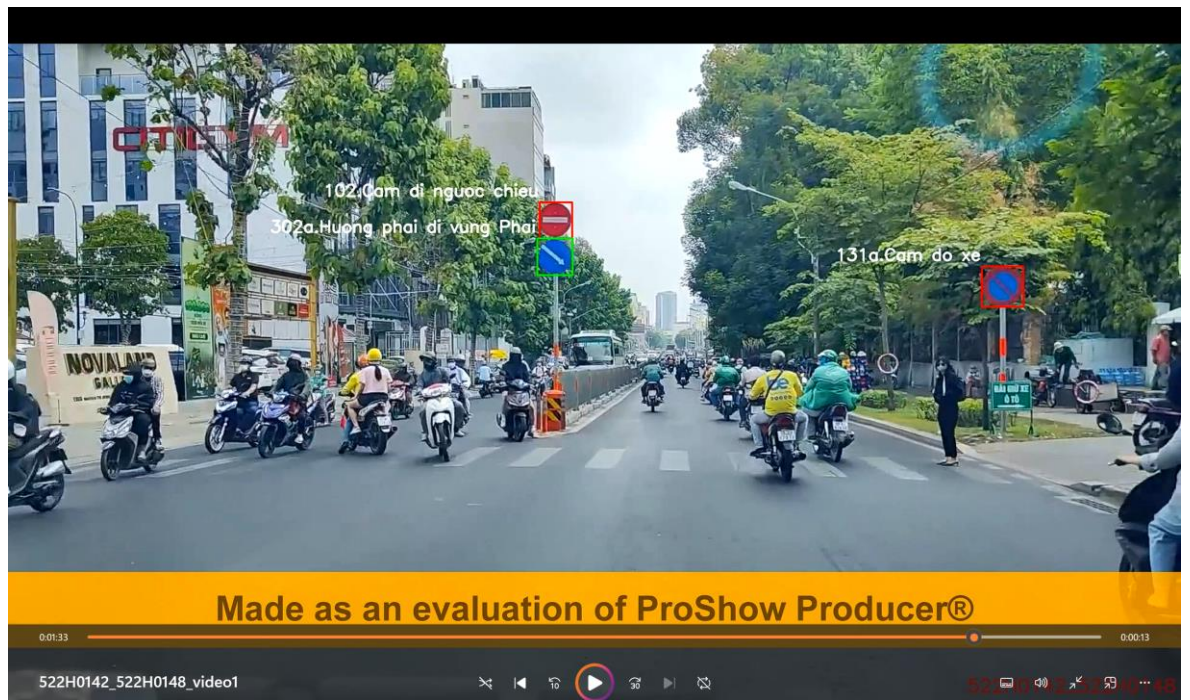


Figure 1.7: Image of video 1

## 2. Video 2:

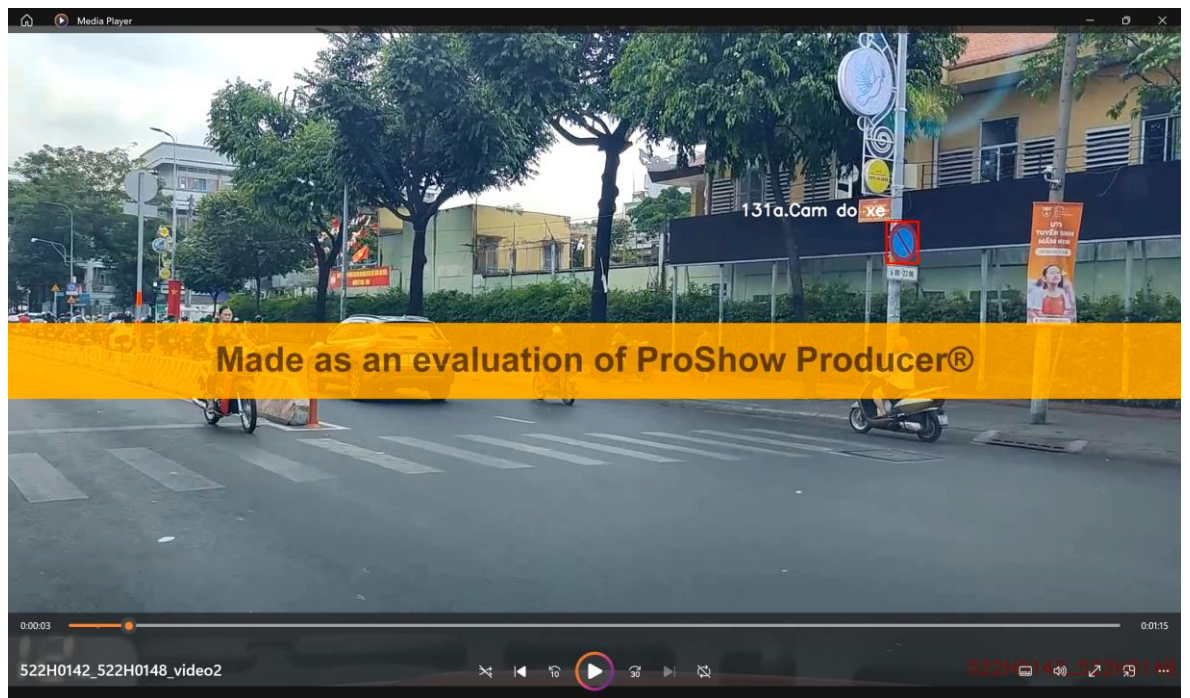


Figure 2.1: Image of video 2



Figure 2.2: Image of video 2



Figure 2.3: Image of video 2





Figure 2.4: Image of video 2

Link output: [https://drive.google.com/drive/folders/1-Re5IZe4XerxZjuSpKMEP1\\_PASP-Rj1-](https://drive.google.com/drive/folders/1-Re5IZe4XerxZjuSpKMEP1_PASP-Rj1-)

## REFERENCES

### LAB 04. MORPHOLOGY IMAGE PROCESSING

<https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/>

<https://www.geeksforgeeks.org/python-opencv-morphological-operations/>

<https://www.freedomvc.com/index.php/2021/06/26/contours-and-bounding-boxes/>

OpenCV: How to define the “lower” and “upper” range of a color?

<https://answers.opencv.org/question/134248/how-to-define-the-lower-and-upper-range-of-a-color/>

<https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/>