VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

MID-TERM PROJECT

INTRODUCTION TO DIGITAL IMAGE PROCESSING

INTRODUCTION TO DIGITAL IMAGE PROCESSING

Instructor: **PhD TRINH HUNG CUONG**

Implement: **LE TRI – 521H0320**

Class **: 21H50302**

Year **: 25**

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

MID-TERM PROJECT

INTRODUCTION TO DIGITAL IMAGE PROCESSING

INTRODUCTION TO DIGITAL IMAGE PROCESSING

Instructor: **PhD TRINH HUNG CUONG**

Implement: **LE TRI – 521H0320**

Class **: 21H50302**

Year **: 25**

HO CHI MINH CITY, 2024

ACKNOWLEDGEMENT

I would like to express our sincere thanks to the Faculty of Information Technology for allowing me to carry out this thesis. At the same time, I would like to express our deep gratitude to Mr. Trinh Hung Cuong for guiding me during the time of studying Digital Image Processing and providing enough knowledge for me to make this report.

Ho Chi Minh city, day , month , year
Author
(Signature and full name)

Le Tri

THIS PROJECT WAS COMPLETED AT TON DUC THANG UNIVERSITY

I hereby declare that this is my 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, I'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, day , month , year

Author

(Signature and full name)

Le Tri

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

Ho Chi Minh November, 2024(Sign and write full name)

Evaluation section for grading instructor

Ho Chi Minh November 2024
(Sign and write full name)

SUMMARY

This report presents a comprehensive analysis and application of digital image processing techniques using OpenCV and NumPy libraries. The project consists of two main exercises focusing on video and image processing. The first exercise involves processing video data to detect traffic signs by applying methods such as color space transformation, Gaussian blur, and contour detection to identify circular and rectangular shapes representing signs. Key steps include configuring video input/output, balancing lighting, creating color thresholds, and filtering noise to enhance object detection accuracy.

The second exercise centers on processing static images by splitting them into distinct sections, converting them to grayscale, and utilizing binary thresholding along with morphological operations to isolate and highlight objects or numbers. Through a structured approach involving adaptive thresholding, contour detection, and customized image processing functions, this project achieves clear object delineation in both dynamic and static image contexts.

This project demonstrates practical skills in digital image processing by implementing algorithms to enhance object detection accuracy, a fundamental aspect of computer vision applications.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

CHAPTER 1 - METHODOLOGY OF SOLVING TASKS

1.1 OpenCV Library

OpenCV (Open Source Computer Vision Library) is an open-source library that is widely used in computer vision and image processing applications. It provides a variety of supporting functions in image recognition and analysis, such as face detection, object recognition, image segmentation, and video processing.

1.2 NumPy Library

NumPy is a powerful Python library for arrays and scientific computing. NumPy provides n-dimensional array objects (ndarrays) and advanced computational tools such as linear algebra, Fourier analysis, and matrix processing. This library is extremely important in processing and analyzing image data when combined with OpenCV.

1.3 Combining OpenCV and NumPy

When using OpenCV in combination with NumPy, image data can be converted into NumPy arrays for quick manipulation and processing. This helps to increase processing speed and performance, especially in tasks that require real-time image and video analysis.

1.3.1 Grayscale

Converting photos to grayscale reduces the complexity of image processing, as each pixel will contain only one luminance value instead of three color (RGB) values. This method is often used in boundary detection, segmentation and object recognition problems

1.3.2 Thresholding

Thresholdization is an image processing technique in which pixels with a luminosity value above or below a certain threshold will be assigned to white or black. This method helps clarify important features in the image, such as detecting objects or distinguishing between backgrounds and objects.

1.3.3 Contour Detection (Contours)

Contour detection is the process of finding the edges of objects in an image. These contours are useful in shape recognition, characterization of objects, and in applications such as handwriting recognition or traffic sign detection.

1.3.4 Bitwise Math

Bitwise math helps to perform bit-level operations between binaries. It is an important tool for performing operations such as extracting selections from photos, masking objects, or combining different parts of a photo.

1.3.5 Morphological Operations

Geometric operations such as dilation, erosion, opening, and closing are often used to process the geometric features of an image. They help eliminate interference, smooth out shapes, or combine small areas together.

1.3.6 Video Processing and Signage Detection

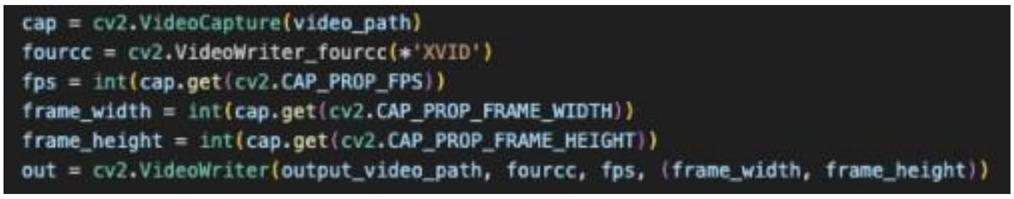
Video processing in OpenCV allows tracking and detection of moving objects in successive frames. In the case of this exercise, the video is processed to detect traffic signs based on their shape and color characteristics, especially those that are rectangular in width rather than height.

1.4 Description of the solving methods

1.4.1 Exercise 1

This code uses OpenCV to process video and detect traffic signs by applying light balance, color transition, and object detection across color thresholds.

Step 1: Configure Video Input and Output

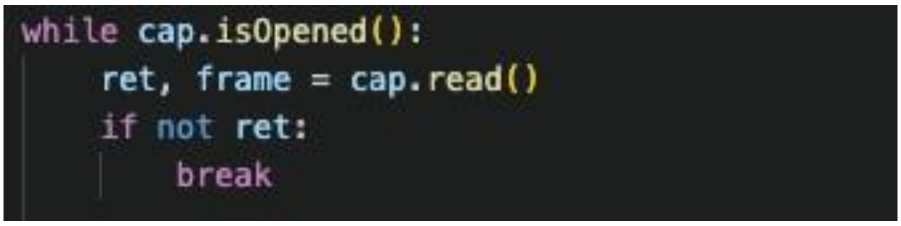


```
cap = cv2.VideoCapture(video_path)
fourcc = cv2.VideoWriter_fourcc(*'XVID')
fps = int(cap.get(cv2.CAP_PROP_FPS))
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width, frame_height))
```

Figure 1

- **CV2. VideoCapture(video_path):** initializes a cap object to read the video from the video_path path. This object will allow each frame of the video to be retrieved during processing
- **CV2. VideoWriter_fourcc(*'XVID'):** Sets up the output video encoding using XVID. fourcc (Four Character Code) is a compression code to reduce the size of a video file while maintaining quality.
- **CV2. VideoWriter:** initializes an out object to record video to the output_video_path path with the following parameters: fourcc compression format, fps frame rate, and frame size (frame_width, frame_height).

Step 2: Loop reading and processing each frame



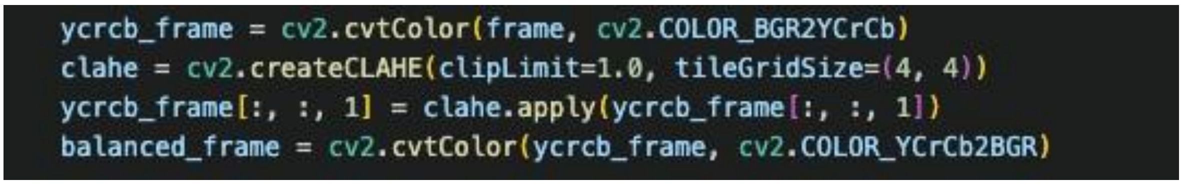
```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
```

Figure 2

- **while cap.isOpened():** Maintains the loop while the video is open.

- **ret, frame = cap.read():** Reads each frame in the video. RET is a boolean value that indicates whether the frame was read successfully.
- **if not ret: break:** Exit the loop when there are no more frames left to read.

Step 3: Balance the Light

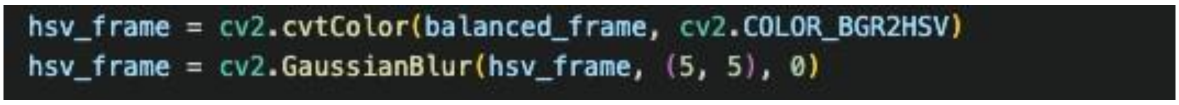


```
ycrcb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(4, 4))
ycrcb_frame[:, :, 1] = clahe.apply(ycrcb_frame[:, :, 1])
balanced_frame = cv2.cvtColor(ycrcb_frame, cv2.COLOR_YCrCb2BGR)
```

Figure 3

- **cv2.cvtColor(frame, cv2. COLOR_BGR2YCrCb):** Converts the frame from the BGR color space to YCrCb, where Y represents brightness.
- **clahe = cv2.createCLAHE(...):** Create a CLAHE object to improve local contrast.
- **ycrcb_frame[:, :, 1] = clahe.apply(ycrcb_frame[:, :, 1]):** Apply CLAHE to balance the light.
- **cv2.cvtColor(ycrcb_frame, cv2. COLOR_YCrCb2BGR):** Converts the frame back to the BGR color space.

Step 4: Convert to HSV space and blur Gaussian

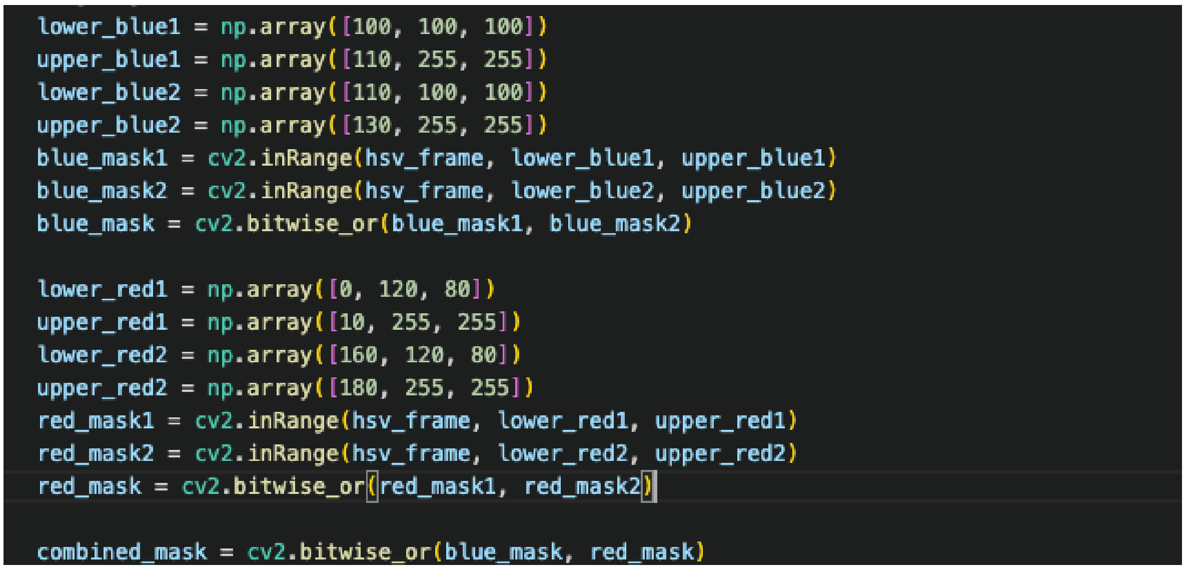


```
hsv_frame = cv2.cvtColor(balanced_frame, cv2.COLOR_BGR2HSV)
hsv_frame = cv2.GaussianBlur(hsv_frame, (5, 5), 0)
```

Figure 4

- **cv2.cvtColor(balanced_frame, cv2. COLOR_BGR2HSV):** Convert the light-balanced frame to the HSV color space.
- **CV2. GaussianBlur(hsv_frame, (5, 5), 0):** Apply Gaussian blur to reduce noise.

Step 5: Apply the color threshold to the blue and red signs



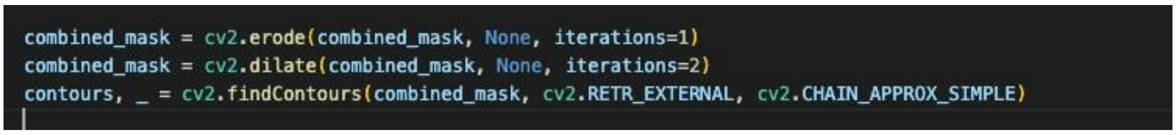
```
lower_blue1 = np.array([100, 100, 100])
upper_blue1 = np.array([110, 255, 255])
lower_blue2 = np.array([110, 100, 100])
upper_blue2 = np.array([130, 255, 255])
blue_mask1 = cv2.inRange(hsv_frame, lower_blue1, upper_blue1)
blue_mask2 = cv2.inRange(hsv_frame, lower_blue2, upper_blue2)
blue_mask = cv2.bitwise_or(blue_mask1, blue_mask2)

lower_red1 = np.array([0, 120, 80])
upper_red1 = np.array([10, 255, 255])
lower_red2 = np.array([160, 120, 80])
upper_red2 = np.array([180, 255, 255])
red_mask1 = cv2.inRange(hsv_frame, lower_red1, upper_red1)
red_mask2 = cv2.inRange(hsv_frame, lower_red2, upper_red2)
red_mask = cv2.bitwise_or(red_mask1, red_mask2)

combined_mask = cv2.bitwise_or(blue_mask, red_mask)
```

Figure 5

• **cv2.inRange(...):** Creates a binary mask for blue and red based on defined color thresholds.

• **cv2.bitwise_or(...):** Combine masks to create a composite mask (combined_mask) for both sign colors.

Step 6: Process and find the contours of the sign



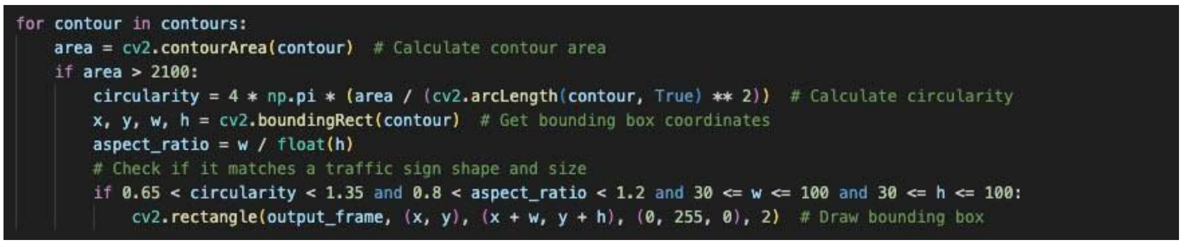
```
combined_mask = cv2.erode(combined_mask, None, iterations=1)
combined_mask = cv2.dilate(combined_mask, None, iterations=2)
contours, _ = cv2.findContours(combined_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Figure 6

• **cv2.erode and cv2.dilate:** Shrink and zoom in on the mask to smooth out the noisy areas.

• **cv2.findContours(...):** Finds contours on the treated mask, representing areas that may be traffic signs.

Step 7: Check the size and shape of the sign



```
for contour in contours:
    area = cv2.contourArea(contour)  # Calculate contour area
    if area > 2100:
        circularity = 4 * np.pi * (area / (cv2.arcLength(contour, True) ** 2))  # Calculate circularity
        x, y, w, h = cv2.boundingRect(contour)  # Get bounding box coordinates
        aspect_ratio = w / float(h)
        # Check if it matches a traffic sign shape and size
        if 0.65 < circularity < 1.35 and 0.8 < aspect_ratio < 1.2 and 30 <= w <= 100 and 30 <= h <= 100:
            cv2.rectangle(output_frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  # Draw bounding box
```

Figure 7

• **cv2.contourArea(contour):** Calculate the area of the contour. This area is used to determine the size of the object, ensuring that the object is large enough to be considered a sign.
• **4 * np.pi * (area / (cv2.arcLength(contour, True) ** 2)):** Calculate the roundness of the object. This formula uses the area and circumference (in cv2.arcLength) to determine the roundness of the object. If the roundness is close to 1, the object may be round.

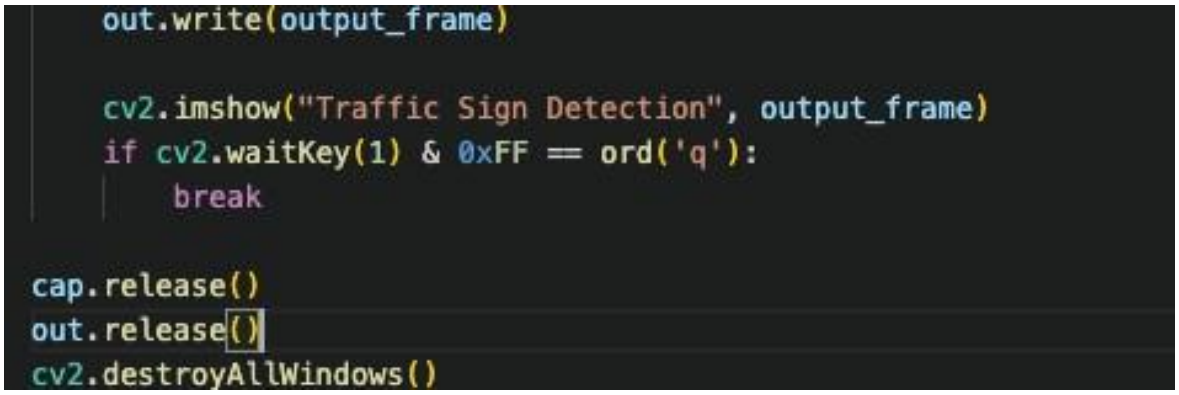
• **cv2.boundingRect(contour):** Computes a rectangle that surrounds the contour of the object. It returns the parameters (x, y) (upper left corner coordinates) and (w, h) (width and height of the rectangle).

• **if ...:** This condition checks three factors to determine if the object is a circular sign:

- **Area (area > 1200):** Consider only objects larger than 1200 pixels.
- **Roundness (0.65 < circularity < 1.35):** The roundness must be within a reasonable range to exclude objects with non-circular shapes.
- **Envelope Ratio (0.8 < aspect_ratio < 1.2): The** ratio between the width and height of the envelope frame must be approximately 1, i.e., the object is almost square or round.
- **Dimensions (30 <= w <= 100 and 30 <= h <= 100): The** size of the envelope frame must be within a certain range to exclude objects that are too small or too large.

• **cv2.rectangle(output_frame, (x, y), (x + w, y + h), (0, 255, 0), 2):** If the above conditions are satisfied, draw a green rectangle ((0, 255, 0)) around the detection object.

Step 8: Record and display the result frame, free up resources



```
    out.write(output_frame)

    cv2.imshow("Traffic Sign Detection", output_frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

Figure 8

- **out.write(output_frame):** Writes the processed frame to the output video.
- **cv2.imshow(...):** Displays the frame in the window.
- **cv2.waitKey(1):** Wait for a short period of time and check if the q key is pressed to exit.
- Free up video resources and close the display window when finished.

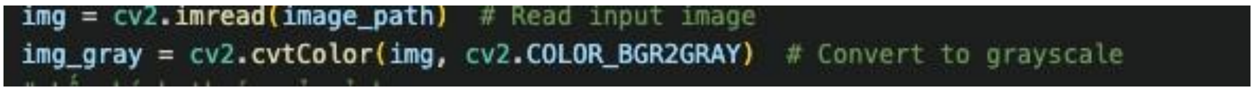
1.4.2 Exercise 2

1.4.2.1 Short description of the main steps of the task

- **The goal of Task 2:** Separate the input image into separate parts (top, bottom left, and bottom right), and then process each part to detect objects or digits using threshold methods and morphological operations.
- **Main steps**:
  1. Read the input image and convert it to grayscale.
  2. Divide the grayscale into sections (top and bottom).
  3. Divide the bottom into two smaller parts, with specific proportions.
  4. Apply image processing methods to each section to detect and draw rectangular boxes around objects.
  5. Combine the reprocessed parts into a single image and display or save the result.

1.4.2.2 Detailed Explanation of Each Programming Statement

Step 1: Read and convert input images:

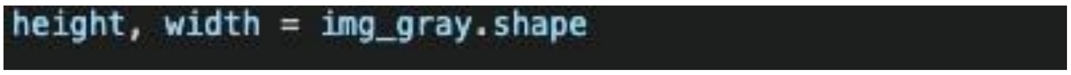


```
img = cv2.imread(image_path)  # Read input image
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
```

Figure 9

- **cv2.imread(image_path):** reads the image from the image_path path and saves the image as a matrix.
- **cv2.cvtColor(img,cv2. COLOR_BGR2GRAY):** converts photos from BGR color space to grayscale to simplify later processing.

Step 2: Get the photo size:

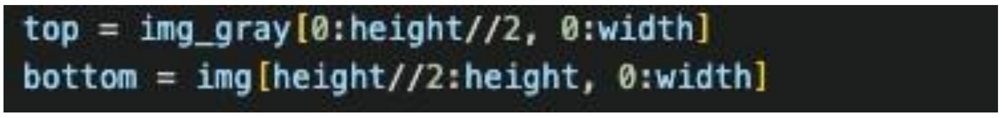


```
height, width = img_gray.shape
```

Figure 10

- Take the height and width of img_gray gray image.

Step 3: Split the photo into upper and lower sections:

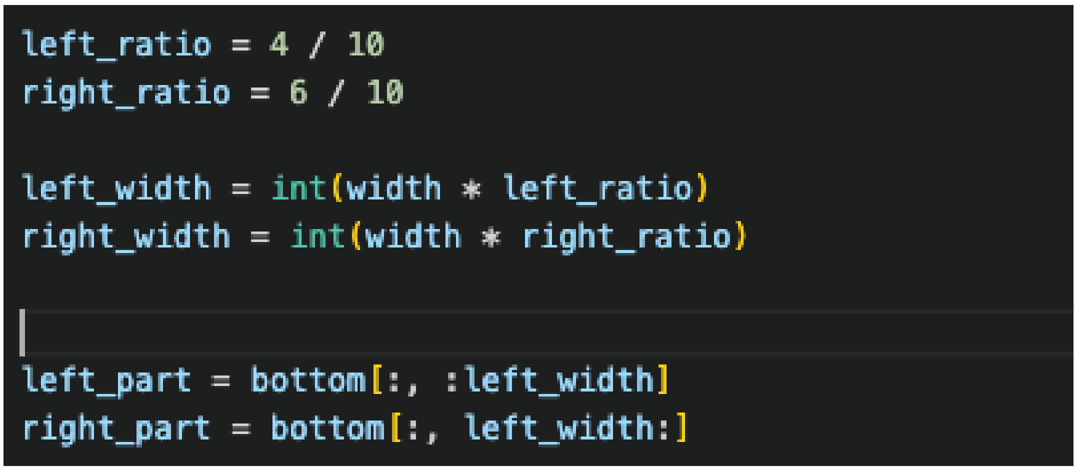


```
top = img_gray[0:height//2, 0:width]
bottom = img[height//2:height, 0:width]
```

Figure 11

- **Explanation**: the top is the top half of the gray image, and the bottom is the bottom half of the image.

Step 4: Calculate the proportions for the sections:

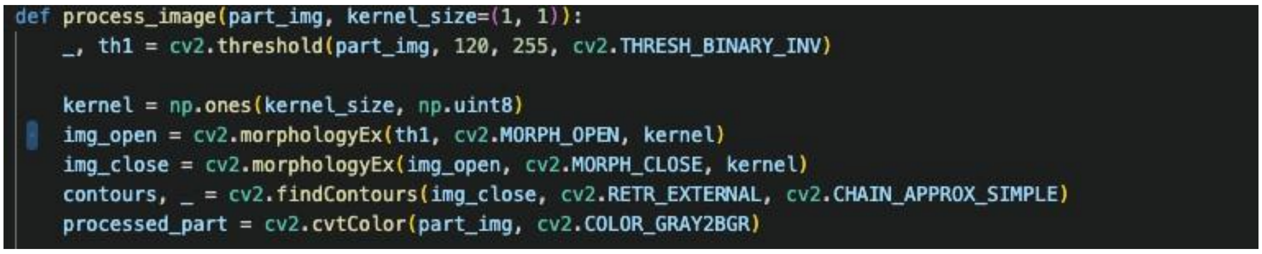


```
left_ratio = 4 / 10
right_ratio = 6 / 10

left_width = int(width * left_ratio)
right_width = int(width * right_ratio)


left_part = bottom[:, :left_width]
right_part = bottom[:, left_width:]
```

Figure 12

- **left_ratio** and **right_ratio** determine the horizontal ratio of the lower part, of which 40% belongs to the left part and 60% to the right part.
- **left_width** and **right_width** are the specific widths of each section.
- **left_part** is the left part (40% width), and **right_part** is the right part (60% width) of the bottom image.

Step 5: Image processing functions with binary and morphological thresholds:

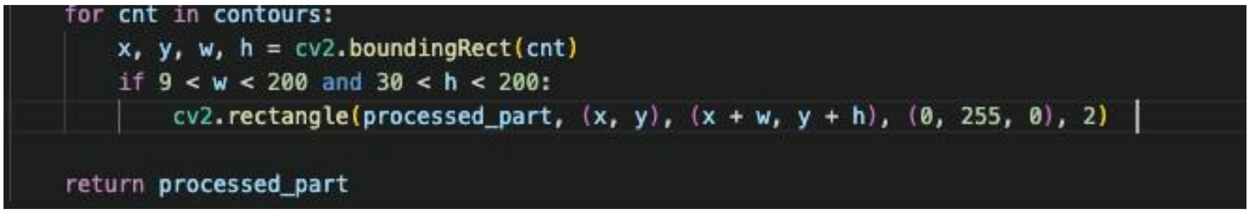


```
def process_image(part_img, kernel_size=(1, 1)):
    _, th1 = cv2.threshold(part_img, 120, 255, cv2.THRESH_BINARY_INV)

    kernel = np.ones(kernel_size, np.uint8)
    img_open = cv2.morphologyEx(th1, cv2.MORPH_OPEN, kernel)
    img_close = cv2.morphologyEx(img_open, cv2.MORPH_CLOSE, kernel)
    contours, _ = cv2.findContours(img_close, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    processed_part = cv2.cvtColor(part_img, cv2.COLOR_GRAY2BGR)
```

Figure 13

- **cv2.threshold(part_img, 120, 255, cv2. THRESH_BINARY_INV)** create an inverted binary image with a threshold of 120.
- **np.ones(kernel_size, np.uint8)** creates a kernel that allows morphological math.
- **cv2.morphologyEx(th1, cv2. MORPH_OPEN, kernel)** applies openness to eliminate interference.
- **cv2.morphologyEx(img_open, cv2. MORPH_CLOSE, kernel)** applies closure to highlight objects.
- **cv2.findContours** finds the contours of the objects after processing.

Step 6: Find and draw a rectangular box around the objects:



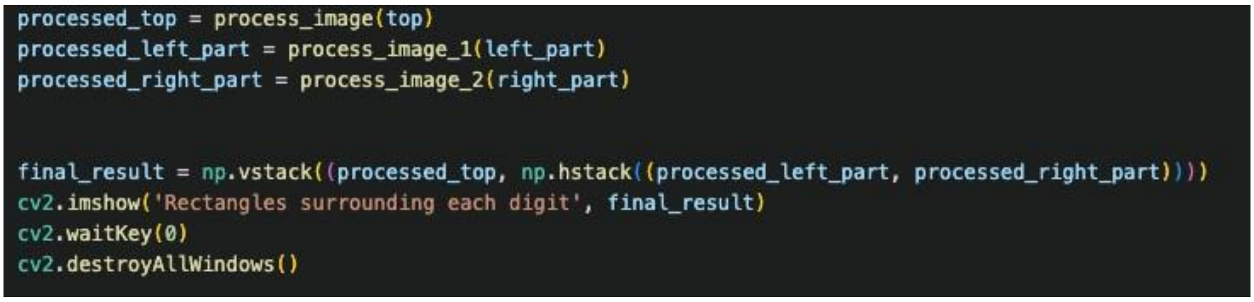
```
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        if 9 < w < 200 and 30 < h < 200:
            cv2.rectangle(processed_part, (x, y), (x + w, y + h), (0, 255, 0), 2)

    return processed_part
```

Figure 14

- The loop for browsing through each contour, calculating the size, and drawing a rectangular box around objects that are between 9 < width < 200 and 30 < height < 200.

Step 7: Combine the parts of the photo and display the result:

- **Explanation**: np.vstack and np.



```
processed_top = process_image(top)
processed_left_part = process_image_1(left_part)
processed_right_part = process_image_2(right_part)


final_result = np.vstack((processed_top, np.hstack((processed_left_part, processed_right_part))))
cv2.imshow('Rectangles surrounding each digit', final_result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 15

- **hstack** stitching the processed parts into one final image. **cv2.imshow** displays the image, and **cv2.imwrite** saves the image to **output_path**.

CHAPTER 2 - TASK RESULTS

2.1 Exercise 1:

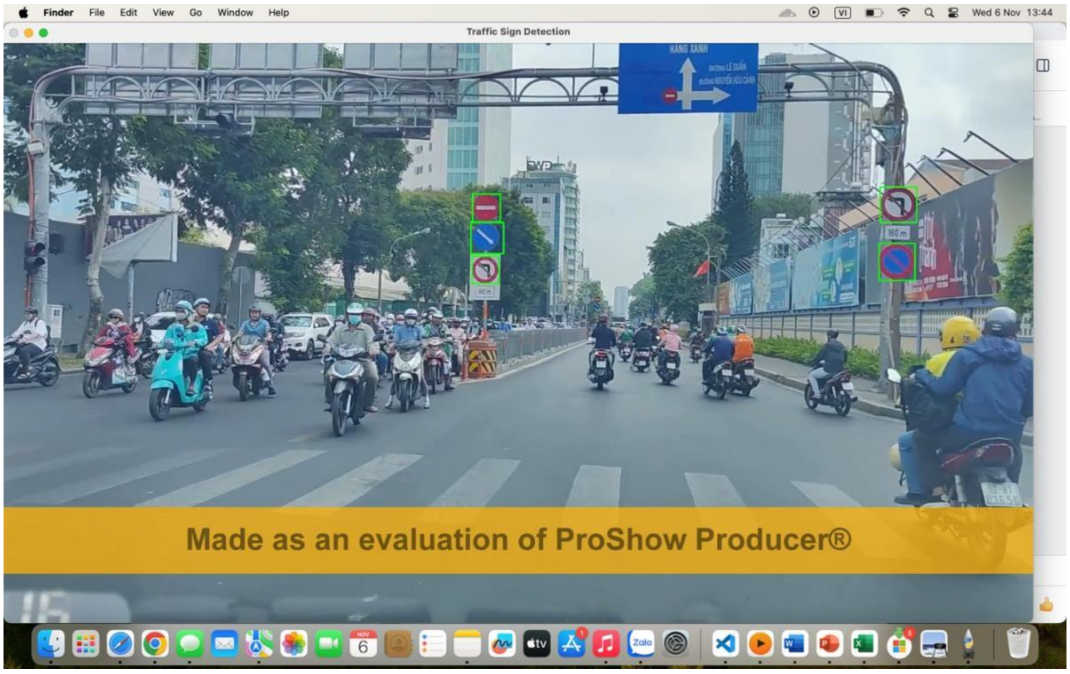

Finder File Edit View Go Window Help
Wed 6 Nov 13:44
Traffic Sign Detection
Made as an evaluation of ProShow Producer®

Figure 16. Output the frame at the 6-second mark.

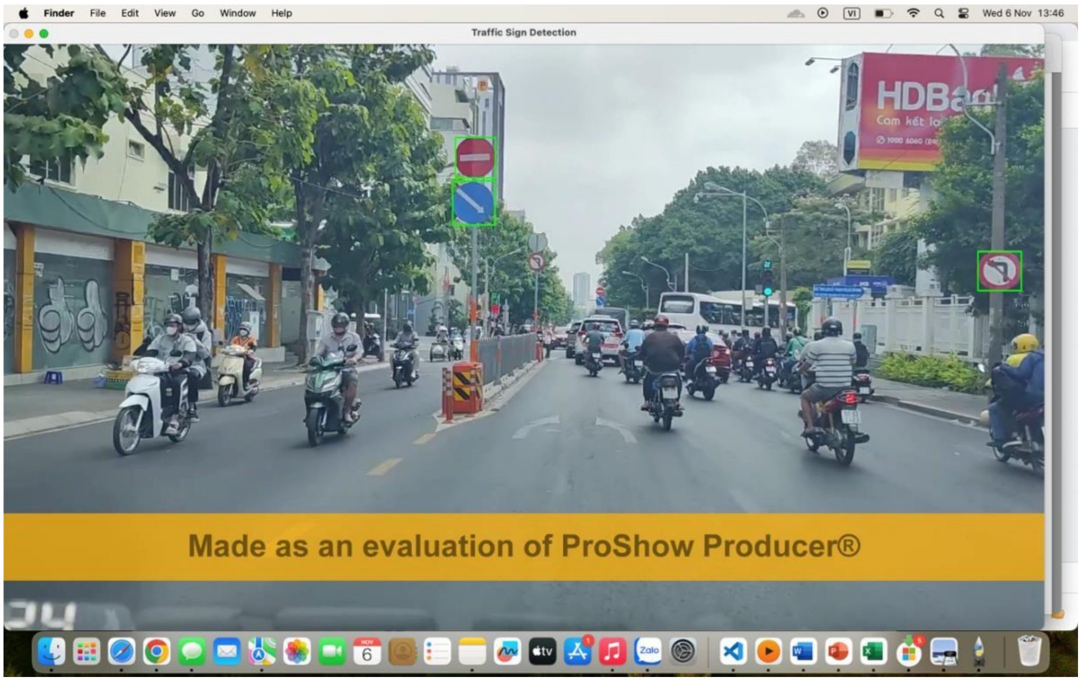

Finder File Edit View Go Window Help
Wed 6 Nov 13:46
Traffic Sign Detection
Made as an evaluation of ProShow Producer®

Figure 17 Output the frame at the 43-second mark.

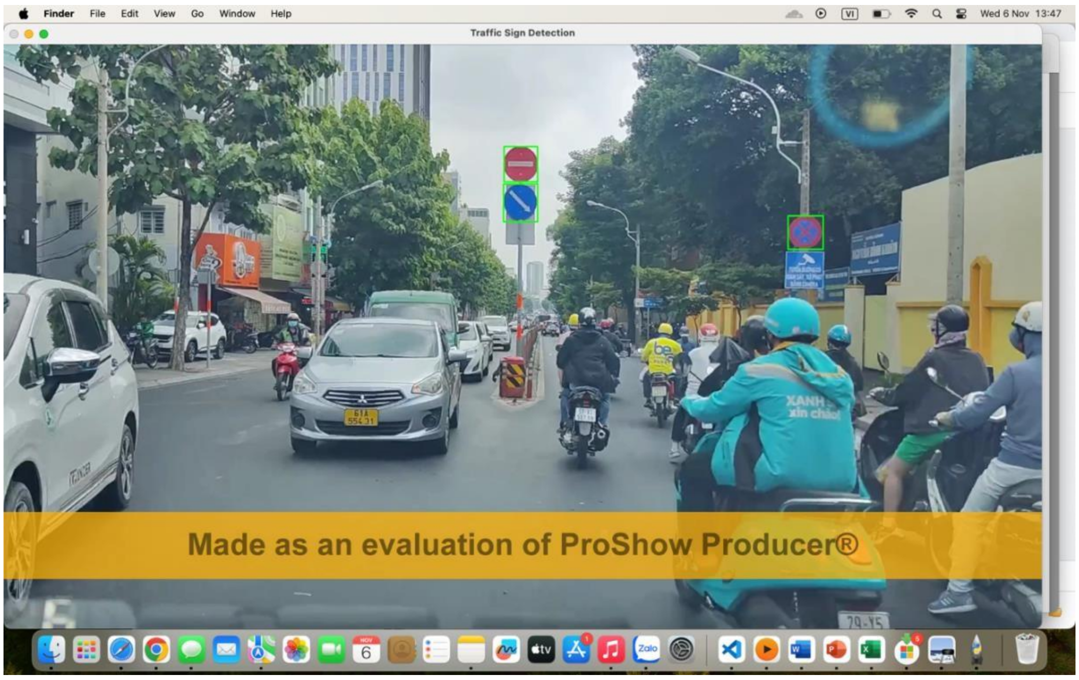

Finder File Edit View Go Window Help
Wed 6 Nov 13:47
Traffic Sign Detection
Made as an evaluation of ProShow Producer®

Figure 18. Output the frame at the 67-second mark.

2.2 Exercise 2

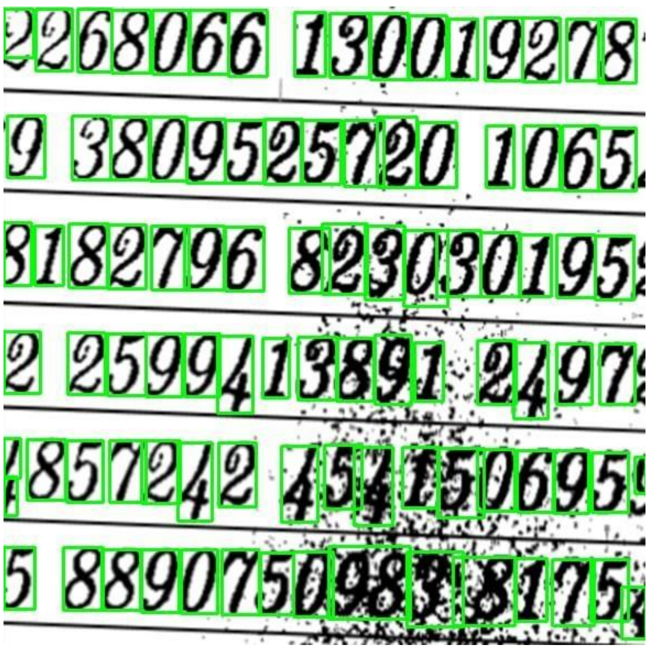


Figure 19. Digits with Bounding Boxes

REFERENCES

https://www.geeksforgeeks.org/opencv-python-tutorial/?ref=shm

https://www.analyticsvidhya.com/blog/2021/05/image-processing-using-opencv-with-practical-examples/

https://learnopencv.com/contour-detection-using-opencv-python-c/

https://drive.google.com/drive/folders/1sYr17cLWyQpVrBgulgHf33Gl_i0n6RN8?dmr=1&ec=wgc-drive-hero-goto