VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

PHAN THÀNH HUY - 521H0244
TRẦN NGỌC PHÁT - 521H0129
LÊ TRÍ - 521H0320

FINAL ESSAY

DIGITAL IMAGE PROCESSING

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

PHAN THÀNH HUY – 521H0244
TRẦN NGỌC PHÁT - 521H0129
LÊ TRÍ - 521H0320

FINAL ESSAY

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 valuable guidance and support throughout the mid-term report of our project on Building a book management and ordering system on the MERN stack platform. He has been very helpful and patient in providing us with constructive feedback and suggestions to improve our work. He has also encouraged us to explore new technologies and techniques to enhance our system's functionality and performance. We have learned a lot from his expertise and experience in web development and software engineering. We are honored and privileged to have him as our teacher and supervisor.

Ho Chi Minh City, 10th December 2024.

Author

(Signature and full name)

<u>Huy</u>

Phan Thành Huy

<u>Phat</u>

Trần Ngọc Phát

<u>Tri</u>

Lê Trí

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.

Ho Chi Minh city, 10th December 2024
Author
(Signature and full name)

Huy

Phan Thành Huy

Phat

Trần Ngọc Phát

Tri

Lê Trí

TABLE OF CONTENTS

INTRODUCTION TO DIGITAL IMAGE PROCESSING

This report outlines the methodology and results for two programming tasks related to image and video processing: detecting traffic signs in a video and detecting digits in an image. The tasks were approached using techniques such as color thresholding, contour detection, and drawing bounding boxes around detected objects, all implemented with OpenCV.

LIST OF FIGURES

CHAPTER 1. METHODOLOGY OF SOLVING TASKS

1.1 Related Background Knowledge and Methods

To detect traffic signs effectively in video frames, the solution employs a combination of digital image processing techniques and methods:

1.1.1 Color Filtering in HSV Color Space

- What is HSV:

- The HSV (Hue, Saturation, Value) color model is more suitable than RGB for detecting specific colors like red, blue, and yellow because it separates chromatic information (color) from intensity.
- **Hue** represents the type of color (red, blue, etc.).
- **Saturation** measures the intensity of the color.
- **Value** measures brightness.

- How HSV is used here:

- Traffic signs are usually in standard red, blue, or yellow colors. The program converts each frame from **BGR** (default OpenCV format) to **HSV** using cv2.cvtColor() and creates binary masks for these colors:

```
lower_red = np.array([150, 50, 50])  # Lower bound of red
upper_red = np.array([180, 255, 255])  # Upper bound of red
# Single blue range
lower_blue = np.array([90, 120, 100])  # Lower bound of blue
upper_blue = np.array([130, 255, 255])  # Upper bound of blue
# Yellow range
lower_yellow = np.array([0, 120, 0])  # Lower bound of yellow
upper_yellow = np.array([30, 255, 255])  # Upper bound of yellow
```

Image 1.1 Color Filtering in HSV Color Space

- A binary mask for each color is created using cv2.inRange(). Pixels within the range are set to white (255), while others are black (0).

1.1.2 Morphological Operations

- **Purpose**: After creating the binary masks, noise can appear as small white regions in the image that do not correspond to traffic signs.
- Steps Taken:
  - **Opening (Noise Removal)**: Removes small white regions caused by noise. This is achieved by first eroding the mask and then dilating it.
  - **Closing (Filling Gaps)**: Closes small gaps within the white regions, ensuring smoother detection.
- **Code Example:**

```
# Noise removal (Morphological operations)
kernel = np.ones((3, 3), np.uint8)
combined_mask = cv2.morphologyEx(combined_mask, cv2.MORPH_OPEN, kernel)
combined_mask = cv2.morphologyEx(combined_mask, cv2.MORPH_CLOSE, kernel)
```

Image 1.2 Noise removal (Morphological operations)

1.1.3 Contour Detection

- **Contours** are continuous curves joining all the points on the boundary of a connected white region in the binary mask.
- **Key Step**:
  - Use cv2.findContours() to detect contours in the combined mask.
  - Filter the contours by area (cv2.contourArea()) to eliminate small regions that are unlikely to be traffic signs.
- **Aspect Ratio Check**:
  - The bounding box around the contour is extracted using cv2.boundingRect(). Only contours with an aspect ratio close to 1:1 (square) are considered valid traffic signs.

1.1.4 Template Matching

- What is Template Matching?
  - Compares a region of interest (ROI) extracted from the frame with preloaded template images of traffic signs.
  - Uses cv2.matchTemplate() to calculate a similarity score between the ROI and each template.
  - The template with the highest similarity score (above a threshold of 0.55) is considered the best match.
- **Steps for Template Matching**:
  - Resize the ROI and template to the same size.
  - Convert both images to grayscale.
  - Use cv2.matchTemplate() with a normalized correlation coefficient (cv2.TM_CCOEFF_NORMED) to calculate the similarity.
  - Similarity threshold: 0.55.

1.1.5 Write the result and display:

- Draw the rectangle and the detected sign name on the frame.
- Write the output video using cv2.VideoWriter().

1.2 Source Code Presentation

1.2.1 Import libraries

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

Image 1.3 Import libraries

- cv2: OpenCV library, used for image and video processing.
- numpy: A Python library for numerical computing, used to handle image data as matrices.

- os: A library to work with the file system, used to read the traffic sign templates from a folder.

1.2.2 Variables and main data structures

1.1.1.1 SIGN_TEMPLATES

```
# Dictionary to store template images
SIGN_TEMPLATES = {}
```

Image 1.4 Dictionary to store template images

1.1.1.2 SIGN_NAME_MAPPING

```
# Dictionary to map template sign names to new names
SIGN_NAME_MAPPING = {
    "camnguocchieu": "Wrong way",
    "wrongway": "Wrong way",
    "huongdi1": "Keep right",
    "keepright": "Keep right",
    "no_left": "No left turn",
    "camdungxe": "No parking",
    "noparking": "No parking",
    "children": "Children",
    "slow": "Slow",
    "nostopandparking": "No stopping and parking",
    "camdungxedonha":"No parking",
    "camdungxe":"No parking",
}
```

Image 1.5 Dictionary to map template sign names to new names

- A dictionary to map the original sign names (from the sign_templates) to more descriptive or user-friendly names.
- **Purpose**: Customize the names of detected signs for display.

1.2.3 load_templates() function

```python
def load_templates(template_dir='sign_templates'):
    """Load template images from directory"""
    for filename in os.listdir(template_dir):
        if filename.endswith(('.png', '.jpg')):
            sign_name = os.path.splitext(filename)[0]
            template_path = os.path.join(template_dir, filename)
            template = cv2.imread(template_path)
            SIGN_TEMPLATES[sign_name] = template
```

Image 1.6 The load_templates function

Functionality:

- Load all traffic sign templates from the folder sign_templates into the SIGN_TEMPLATES dictionary.

Steps:

1. **Iterate through files**: Use os.listdir(template_dir) to list all files in the directory.
2. **Filter image files**: Only process files with extensions .png or .jpg.
3. **Load images**: Use cv2.imread(template_path) to load each template image.
4. **Store templates**: Save each template into the dictionary SIGN_TEMPLATES with the filename (minus the extension) as the key.

1.2.4 match_template() function

```
def match_template(roi, template, target_size=(100, 100)):
    """Compare ROI with template using template matching"""
    # Resize both images to same size
    roi_resized = cv2.resize(roi, target_size)
    template_resized = cv2.resize(template, target_size)

    # Convert both to grayscale
    roi_gray = cv2.cvtColor(roi_resized, cv2.COLOR_BGR2GRAY)
    template_gray = cv2.cvtColor(template_resized, cv2.COLOR_BGR2GRAY)

    # Template matching
    result = cv2.matchTemplate(roi_gray, template_gray, cv2.TM_CCOEFF_NORMED)
    return np.max(result)
```

Image 1.7 The match_template function

Functionality:

- Compare a region of interest (ROI) in the frame with a traffic sign template using **template matching**.

Steps:

1. **Resize images:** Normalize the ROI and template to a consistent size (100x100) using cv2.resize.
2. **Convert to grayscale:** Simplify the images by converting them to grayscale using cv2.cvtColor.
3. **Perform template matching:** Use cv2.matchTemplate to calculate a similarity score between the ROI and the template.
4. **Return maximum score:** Extract the highest similarity score using np.max(result).

1.2.5 detect_traffic_signs() function

```python
def detect_traffic_signs(frame):
    # Convert to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Define color ranges
    # Single red range
    lower_red = np.array([150, 50, 50])  # Lower bound of red
    upper_red = np.array([180, 255, 255])  # Upper bound of red
    # Single blue range
    lower_blue = np.array([90, 120, 100])  # Lower bound of blue
    upper_blue = np.array([130, 255, 255])  # Upper bound of blue
    # Yellow range
    lower_yellow = np.array([0, 120, 0])  # Lower bound of yellow
    upper_yellow = np.array([30, 255, 255])  # Upper bound of yellow

    # Create red, blue, and yellow masks
    red_mask = cv2.inRange(hsv, lower_red, upper_red)
    blue_mask = cv2.inRange(hsv, lower_blue, upper_blue)
    yellow_mask = cv2.inRange(hsv, lower_yellow, upper_yellow)

    # Combine the red, blue, and yellow masks
    combined_mask = cv2.bitwise_or(red_mask, blue_mask)
    combined_mask = cv2.bitwise_or(combined_mask, yellow_mask)

    # Noise removal (Morphological operations)
    kernel = np.ones((3, 3), np.uint8)
    combined_mask = cv2.morphologyEx(combined_mask, cv2.MORPH_OPEN, kernel)
    combined_mask = cv2.morphologyEx(combined_mask, cv2.MORPH_CLOSE, kernel)

    # Find contours
    contours, _ = cv2.findContours(combined_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    detected_signs = []

    for contour in contours:
        if cv2.contourArea(contour) > 500:
            x, y, w, h = cv2.boundingRect(contour)
            aspect_ratio = float(w) / h

            if 0.8 <= aspect_ratio <= 1.5:
                # Extract ROI
                roi = frame[y:y + h, x:x + w]

                # Get color type (Red, Blue, or Yellow)
                roi_hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
                red_pixels = cv2.countNonZero(cv2.inRange(roi_hsv, lower_red, upper_red))
                blue_pixels = cv2.countNonZero(cv2.inRange(roi_hsv, lower_blue, upper_blue))
                yellow_pixels = cv2.countNonZero(cv2.inRange(roi_hsv,
lower_yellow,upper_yellow))

                # Determine the color type
                if red_pixels > max(blue_pixels, yellow_pixels):
                    color_type = "Red"
                elif blue_pixels > max(red_pixels, yellow_pixels):
                    color_type = "Blue"
                else:
                    color_type = "Yellow"

                # Template matching
                best_match = None
                best_score = 0

                for sign_name, template in SIGN_TEMPLATES.items():
                    score = match_template(roi, template)
                    if score > best_score and score > 0.55:
                        best_score = score
                        best_match = sign_name

                if best_match:
                    # Map the sign name to a custom name
                    new_sign_name = SIGN_NAME_MAPPING.get(best_match, best_match)  # If no
mapping, use the original name
                    sign_info = f"{new_sign_name} ({color_type})"
                    detected_signs.append((x, y, w, h, sign_info))

    return detected_signs
```

Image 1.8 The detect_traffic_signs function

Functionality:

- Detect traffic signs in a single frame of the video.

Steps:

1. **Convert frame to HSV color space**: Use cv2.cvtColor to convert the frame from RGB to HSV, which simplifies color-based segmentation.
2. **Define color ranges:**
   a) Create ranges for red, blue, and yellow in HSV color space.
   b) These ranges help identify traffic signs based on their dominant colors.
3. **Create masks:**
   a) Use cv2.inRange to create binary masks for red, blue, and yellow areas.
   b) Combine the masks using cv2.bitwise_or to get a single mask for all traffic signs.
4. **Noise removal:**
   a) Use morphological operations (cv2.morphologyEx) to clean up the mask by removing noise and filling gaps.
5. **Find contours:**
   a) Extract the shapes in the mask using cv2.findContours.
   b) Filter contours based on area (>500 pixels) and aspect ratio (close to square).
6. **Template matching:**
   a) For each ROI (Region of Interest) corresponding to a detected contour:
      i. Compare it with all templates in SIGN_TEMPLATES using match_template.
      ii. Keep track of the best matching template if the score exceeds 0.55.
7. **Color classification:**
   a) Check the dominant color (red, blue, or yellow) in the ROI by counting the number of pixels that match each color range.
8. **Store results:**
   a) Append the detected sign's position and name to a list.

1.2.6 process_video() function

```python
def process_video(video_path):
    # Load template images
    load_templates()

    cap = cv2.VideoCapture(video_path)

    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    out = cv2.VideoWriter('output2.avi',
                          cv2.VideoWriter_fourcc(*'XVID'),
                          fps, (frame_width, frame_height))

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

        signs = detect_traffic_signs(frame)

        for (x, y, w, h, sign_name) in signs:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            cv2.putText(frame, sign_name, (x, y-10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

        cv2.imshow("Processed Video", frame)
        out.write(frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

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

Image 1.9 The detect_traffic_signs function

Functionality:

- Process the input video to detect traffic signs in each frame, annotate them, and save the result as a new video.

Steps:

1. **Load templates**: Call load_templates() to load traffic sign templates into memory.
2. **Read input video**:
   a) Use cv2.VideoCapture to load the video.
   b) Extract video properties such as frame width, height, and FPS.
3. **Prepare output video**:
   a) Create a cv2.VideoWriter object to save the processed frames into a new video.
4. **Process frames**:
   a) For each frame:
      i. Call detect_traffic_signs() to detect traffic signs.
      ii. Draw bounding boxes and labels for each detected sign using cv2.rectangle and cv2.putText.
      iii. Save the annotated frame to the output video.
5. **Release resources**:
   a) Close the video input and output streams and destroy all OpenCV windows.

1.2.7 Directory Structure

```
# Create directory structure
if not os.path.exists('sign_templates'):
    os.makedirs('sign_templates')
```

Image 1.10 Create directory structure

Ensure that the sign_templates folder exists to store traffic sign template images.

1.2.8 Example Usage

```
# Process video
process_video('video2.mp4')
```

Image 1.11 Process video

- Call the process_video() function with the input video (video1 or video2).
- The script processes the video frame by frame, detects traffic signs, and saves the output as output2.avi.

CHAPTER 2. TASK RESULT

2.1 Task result of video 1:

Image 2.1 Output the frame at 5 seconds mark in video 1

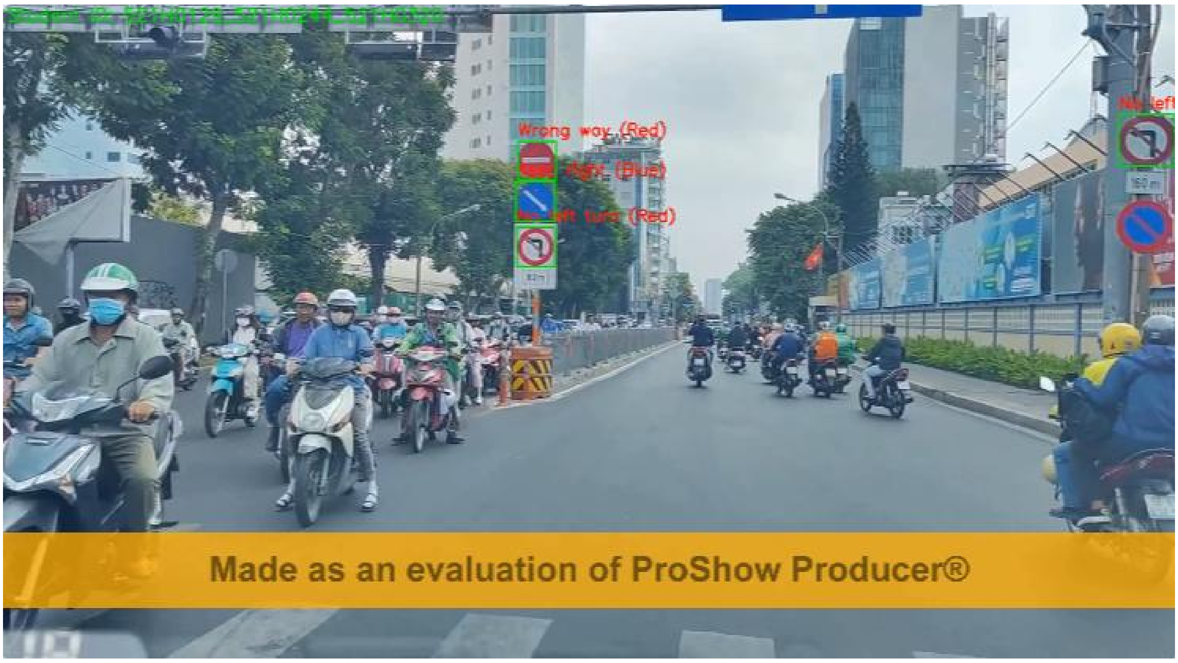

Image 2.2 Output the frame at 7 seconds mark in video 1

Image 2.3 Output the frame at 18 seconds mark in video 1

Image 2.4 Output the frame at 25 seconds mark in video 1

Image 2.5 Output the frame at 36 seconds mark in video 1

Image 2.6 Output the frame at 42 seconds mark in video 1

Image 2.7 Output the frame at 46 seconds mark in video 1

Image 2.8 Output the frame at 57 seconds mark in video 1

Image 2.9 Output the frame at 1 minutes and 9 seconds mark in video 1

Image 2.10 Output the frame at 1 minutes and 26 seconds mark in video 1

Image 2.11 Output the frame at 1 minutes and 46 seconds mark in video 1

2.2 Task result of video 2:

Image 2.12 Output the frame at 4 seconds mark in video 2

Image 2.13 Output the frame at 23 seconds mark in video 2

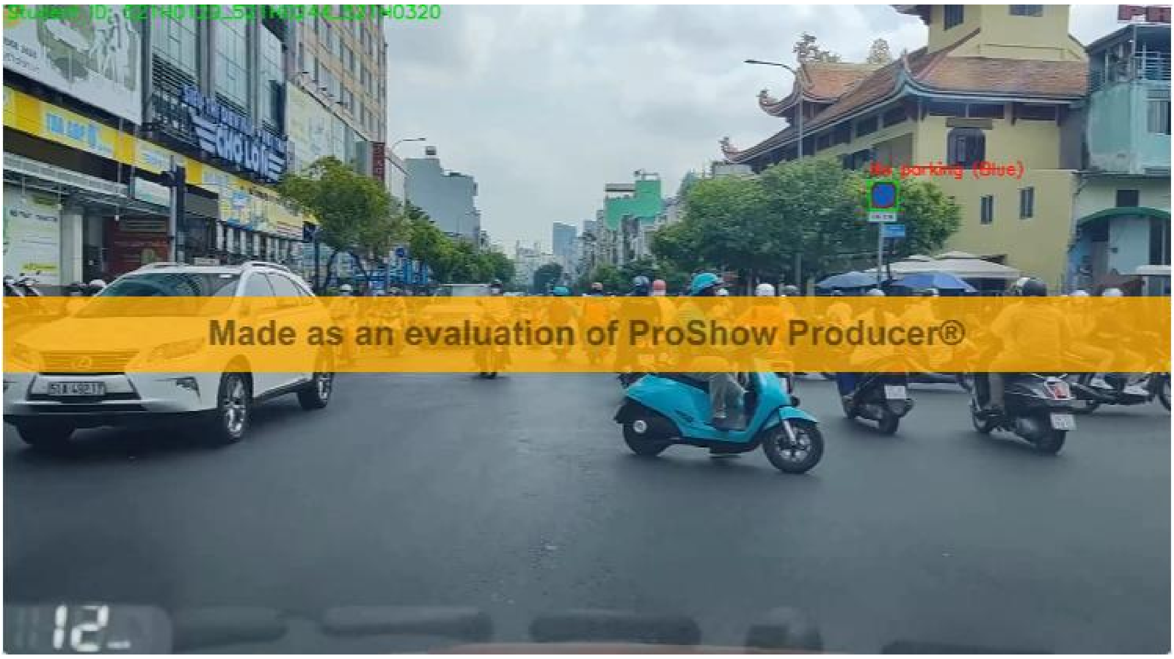

Image 2.14 Output the frame at 1 minutes and 4 seconds mark in video 2

REFERENCES

1. OpenCV Team. OpenCV Documentation. Available at: https://docs.opencv.org
2. Gonzalez, R. C., & Woods, R. E. (2008). Digital Image Processing (3rd ed.). Prentice Hall.
3. Shapiro, L. G., & Stockman, G. C. (2001). Computer Vision. Prentice Hall.
4. Rosebrock, A. PyImageSearch Blog: OpenCV Tutorials and Examples. Available at: https://pyimagesearch.com
5. OpenCV Team. Tutorials in OpenCV: Video Analysis. Available at: https://docs.opencv.org/master/dd/d43/tutorial_py_video_display.html
6. Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.