EMBEDDED SYSTEM AND IOT DESIGN

PROJECT REPORT

VEHICLE NUMBER PLATE READING

BATCH NO: 03

810021106043 - KRISHNAKUMAR C

810021106044 - KUMARAVEL M

810021106045 - LAVANYA S

810021106712 - RAGHUNATHRAGHAV S

INTRODUCTION:

Characters reading for number plates, also known as Optical Character Recognition (OCR). In today's world, automated systems for recognizing vehicle plate numbers play a vital role in various applications, including traffic management ,parking systems, and law enforcement. Python, with its rich libraries for image processing and machine learning, offers a robust platform for developing such systems.

OBJECTIVES:

The objective of this project is to develop a Python script capable of reading and extracting vehicle plate numbers from images or video streams.

KEY COMPONENTS:

- Image acquisition
- Pre processing
- Plate detection
- Character segmentation
- Optical character recognition
- Post processing
- output

LITERATURE SURVEY:

Various image processing techniques are employed for vehicle plate detection, segmentation, and recognition. These techniques include edge detection, contour analysis, morphological operations, and template matching. OCR algorithms play a crucial role in recognizing characters from license plates. Techniques range from traditional methods like template matching to more advanced neural network-based approaches. Datasets containing annotated images of vehicle plates are essential for training and evaluating ALPR systems. Several benchmark datasets have been created for this purpose, aiding in the development and comparison of different algorithms.

SOFTWARE:

Software used: Python (3.12.2), Tesseract-OCR (by google)

OpenCV (Open Source Computer Vision Library): OpenCV is a popular open-source computer vision and machine learning software library. It provides various functions for image processing, computer vision tasks, and video analysis.

NumPy: NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. In the code, NumPy is utilized for array manipulation and mathematical operations, especially in conjunction with OpenCV.

Pytesseract: Pytesseract is a Python wrapper for Google's Tesseract-OCR Engine. It allows easy integration of Tesseract into Python applications, enabling optical character recognition (OCR) from images. The code uses Pytesseract to perform OCR on the extracted plate region and recognize the text.

Tesseract-OCR Engine: Tesseract is an open-source OCR engine maintained by Google. It is widely used for text recognition from images and is known for its accuracy and language support. Pytesseract interacts with the Tesseract-OCR Engine to perform text recognition tasks.

Python Programming Language: Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used in various fields, including data science, machine learning, web

development, and scientific computing. The entire program is written in Python, making use of its syntax, libraries, and ecosystem for image processing and OCR tasks.

PROGRAM:

```
import cv2

import numpy as np

import pytesseract

pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

def preprocess_image(image_path):

    # Read the image

    img = cv2.imread(image_path)

    # Convert to grayscale

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur

    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # Apply thresholding

    _, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    return thresh

def find_contours(thresh):

    # Find contours

    contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Sort contours based on their area

    contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]

    return contours

def extract_plate(contours, image):
```

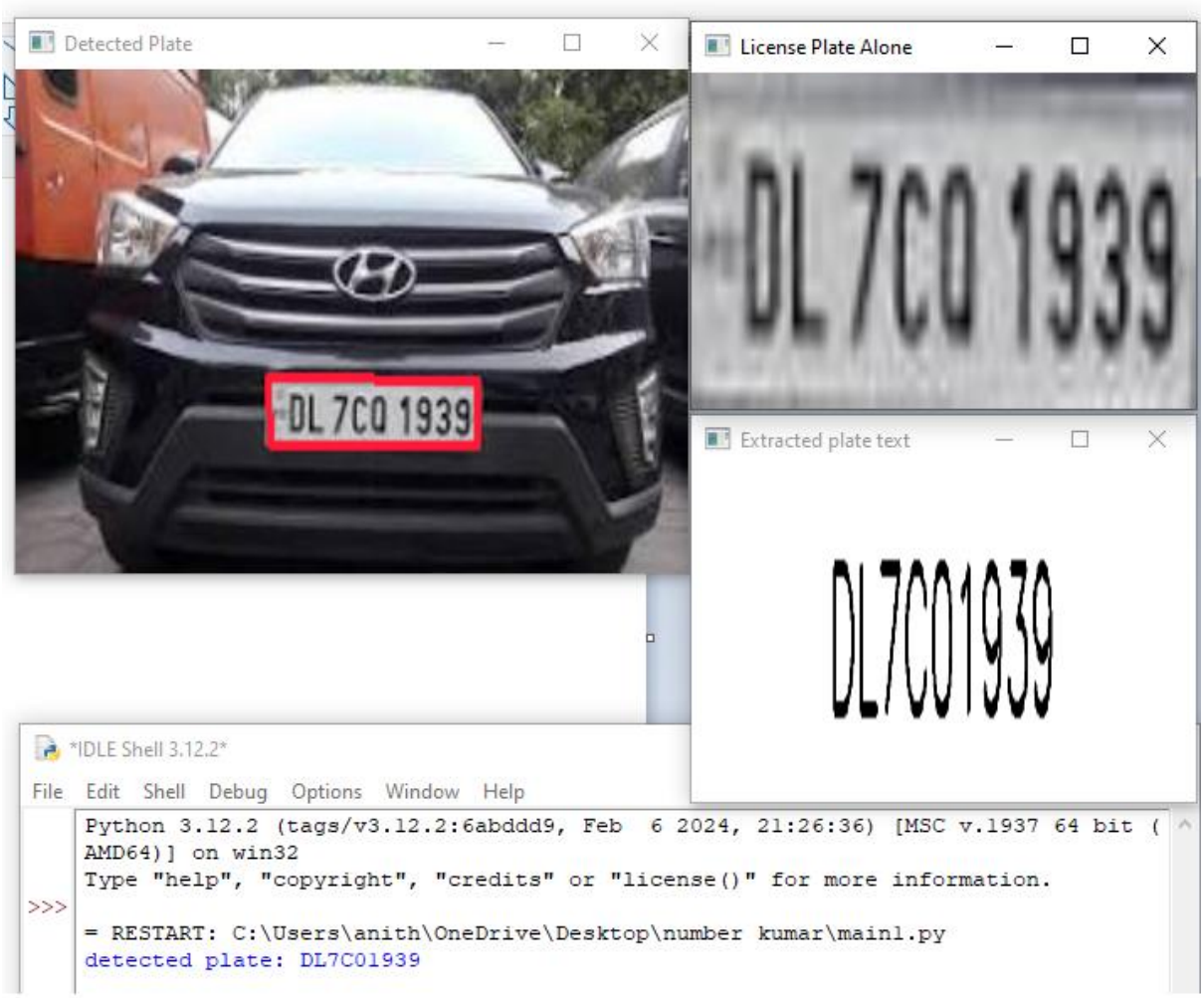
```
    # Initialize an empty plate location
    plate = None
    for contour in contours:
        # Get approximate polygon of the contour
        perimeter = cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, 0.02 * perimeter, True)
        # If the contour has 4 vertices, it is likely a plate
        if len(approx) == 4:
            plate = approx
            break
    if plate is not None:
        # Draw the plate on the original image
        cv2.drawContours(image, [plate], -1, (54, 25, 342), 2)
    return plate, image
def recognize_plate(plate, image):
    # Mask the plate region
    mask = np.zeros(image.shape, dtype=np.uint8)
    cv2.drawContours(mask, [plate], -1, (255, 255, 255), -1)
    masked_img = cv2.bitwise_and(image, mask)
    # Extract plate region
    (x, y) = np.where(mask[:, :, 1] == 255)
    (topx, topy) = (np.min(x), np.min(y))
    (bottomx, bottomy) = (np.max(x), np.max(y))
    cropped_img = masked_img[topx:bottomx+1, topy:bottomy+1]
    # Use pytesseract to extract text from plate region
    plate_text = pytesseract.image_to_string(cropped_img, config='--psm 8')
```

```
    return plate_text
def main(image_path):
    # Preprocess image
    thresh = preprocess_image(image_path)
    # Find contours
    contours = find_contours(thresh.copy())
    # Read the original image
    image = cv2.imread(image_path)
    # Extract plate
    plate, image_with_plate = extract_plate(contours, image.copy())
    if plate is not None:
        # Recognize plate
        plate_text = recognize_plate(plate, image)
        # Display the plate image
        cv2.imshow("Detected Plate", cv2.resize(image_with_plate, (400, 300)))  # Resize and display plate image
        # Extract and display license plate alone
        (x, y, w, h) = cv2.boundingRect(plate)
        plate_alone = image[y:y + h, x:x + w]
        cv2.imshow("License Plate Alone", cv2.resize(plate_alone, (300, 200)))  # Resize and display plate alone image
        cv2.imwrite("license_plate_alone.jpg", plate_alone)  # Save license plate alone as an image file
        # Extract characters and integers from the extracted text
        plate_text_filtered = ''.join(char for char in plate_text if char.isalnum())
        print("detected plate:", plate_text_filtered)
        # Display the extracted text
```

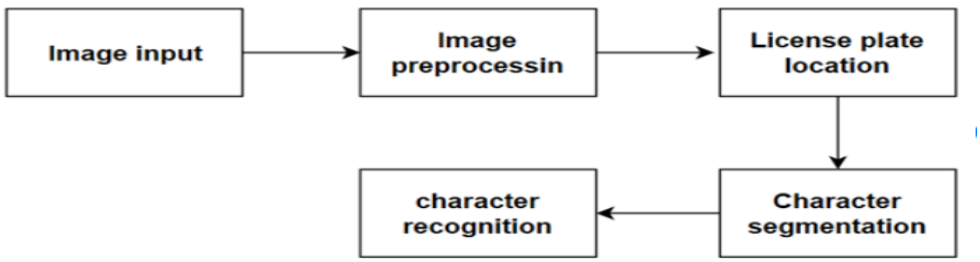
```
        text_img = np.ones((100, 800, 3), dtype=np.uint8) * 255  # Create a blank image for text display
        font_scale = 2
        font_thickness = 3
        font = cv2.FONT_HERSHEY_SIMPLEX
        text_size = cv2.getTextSize(plate_text_filtered, font, font_scale, font_thickness)[0]
        text_x = max((text_img.shape[1] - text_size[0]) // 2, 0)
        text_y = max((text_img.shape[0] + text_size[1]) // 2, 0)
        cv2.putText(text_img, plate_text_filtered, (text_x, text_y), font, font_scale, (0, 0, 0), font_thickness)
        cv2.imshow("Extracted plate text", cv2.resize(text_img, (300, 200)))  # Resize and display filtered plate text
        cv2.imwrite("extracted_text_image.jpg", text_img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print("Plate not found")
if __name__ == "__main__":
    image_path = r"C:\Users\anith\OneDrive\Desktop\number kumar\images\001.jpeg"  # Provide the path to the image containing the car
    main(image_path)
```

RESULT:

OUTPUT:



FLOWCHART:



CONCLUSION:

Advantages:

- It can process a large number of number plates quickly, enabling rapid identification and analysis of vehicles in real-time scenarios. This speed is crucial for applications like traffic monitoring, border control, and access control systems.
- They can be customized and trained to recognize number plates from different regions, countries, and language.
- It generate valuable data insights by capturing and analyzing vehicle-related information. This data can be used for traffic analysis, trend identification, and decision-making in urban planning, transportation management, and law enforcement.

Disadvantages:

- Number plates can vary significantly in format, font style, size, and positioning across different regions and countries. OCR systems may struggle to adapt to these variations, leading to lower recognition accuracy and the need for customizations or region-specific training datasets.
- OCR systems may be vulnerable to manipulation or spoofing attempts, where malicious actors intentionally modify or obscure the number plate to evade detection or falsify information.
- The cost of implementation and maintenance may be a barrier for some organizations, particularly smaller businesses or agencies with limited budgets.

Limitations:

- The accuracy of number plate recognition systems heavily depends on the quality of the input images and the effectiveness of the image processing algorithms used. Poor image quality, such as low resolution or blur, can significantly impact the accuracy of the recognition process.
- Real-time processing of video streams for number plate recognition can be computationally intensive, especially when dealing with high-resolution video feeds or a large number of concurrent streams. This can limit the scalability and speed of the system, especially on hardware with limited processing power.