# Report for Manual Extraction of Sellipi Characters

Team Members

A A Ahamed – KIC-HNDCSAI-Y2-231-F-014

H M B L Herath - KIC-HNDCSAI-Y2-231-F-015

# Table of content

# 1. Introduction

## 1.1 Project Overview

The Manual Extraction of Sellipi Characters project aimed to develop an algorithm capable of extracting and recognizing characters from lnsciption plate images. The project holds significance in various applications.

# 2. Methodology

## 2.1 Image Preprocessing

The project began by loading the image using OpenCV. The team chose a sample inscription plate image ('plate.jpg') for initial testing. Grayscale conversion and Gaussian blur (kernel size of 5x5) were applied to reduce noise and enhance subsequent edge detection. The adaptive thresholding method was employed using `cv2.adaptiveThreshold` with parameters set to (255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 13, 7).

code

```
 Image preprocessing

img = cv2.imread('plate.jpg')

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

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

thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY_INV, 13, 7)
```

## 2.2 Contour Detection

Contours were identified using `cv2.findContours`, and filtering was performed to exclude contours with an area below a specified threshold (`min_contour_area = 300`). The filtered contours were drawn on a blank mask, and dilation was applied using a 5x5 kernel to separate lines.

code

```
 Contour detection and filtering

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

filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) > min_contour_area]

contour_mask = np.zeros_like(gray)

cv2.drawContours(contour_mask, filtered_contours, -1, (255), thickness=cv2.FILLED)

dilated_mask = cv2.dilate(contour_mask, np.ones((5, 5), np.uint8), iterations=1)
```

## 2.3 Dilation and Mask Application

The dilated mask was applied to the original image, creating a new image where only the regions of interest were retained.

code

```
 Dilate the mask and apply to the original image

dilated_img = np.zeros_like(img)

dilated_img[dilated_mask > 0] = img[dilated_mask > 0]
```

## 2.4 Letter Cropping and Display

Individual letters were cropped using bounding rectangles, and conditions were applied to ensure the exclusion of small artifacts.

code

```
 Crop and display each letter
for i, cnt in enumerate(filtered_contours):
    x, y, w, h = cv2.boundingRect(cnt)

     Check conditions for cropping
    if y >= 30 and x >= 30:
        letter_crop = dilated_img[y:y + h, x:x + w]

         Display each cropped letter
        plt.figure(figsize=(2, 2))
        plt.imshow(cv2.cvtColor(letter_crop, cv2.COLOR_BGR2RGB))
        plt.title(f'Letter {i+1}')
        plt.show()
```

# 3. Challenges Faced

## 3.1 Image Variability

One major challenge encountered was the variability in inscriptionplate images. Different lighting conditions, plate sizes, and text styles posed difficulties in creating a one-size-fits-all solution. This led to the need for parameter fine-tuning to handle diverse scenarios effectively.

## 3.2 Noise Reduction

Noise reduction during preprocessing presented a challenge, especially when dealing with images that had inherent noise or artifacts. The team experimented with various blur kernel sizes and thresholding techniques to strike a balance between noise reduction and maintaining crucial details.

## 3.3 Contour Filtering

The contour filtering process faced challenges in distinguishing between relevant contours and potential artifacts. Optimizing the minimum contour area parameter proved crucial in retaining the inscriptionplate characters while eliminating unwanted elements.

# 4. Insights Gained

## 4.1 Algorithm Performance

The character recognition algorithm exhibited promising results in successfully isolating and displaying inscriptionplate characters. However, it became apparent that further refinement would be necessary to enhance performance under a wider range of conditions.

## 4.2 Variability Handling

Insights gained from handling image variability emphasized the importance of creating a robust and adaptable algorithm. The team recognized the need for a more dynamic approach to parameter selection, potentially incorporating machine learning techniques for better adaptability.

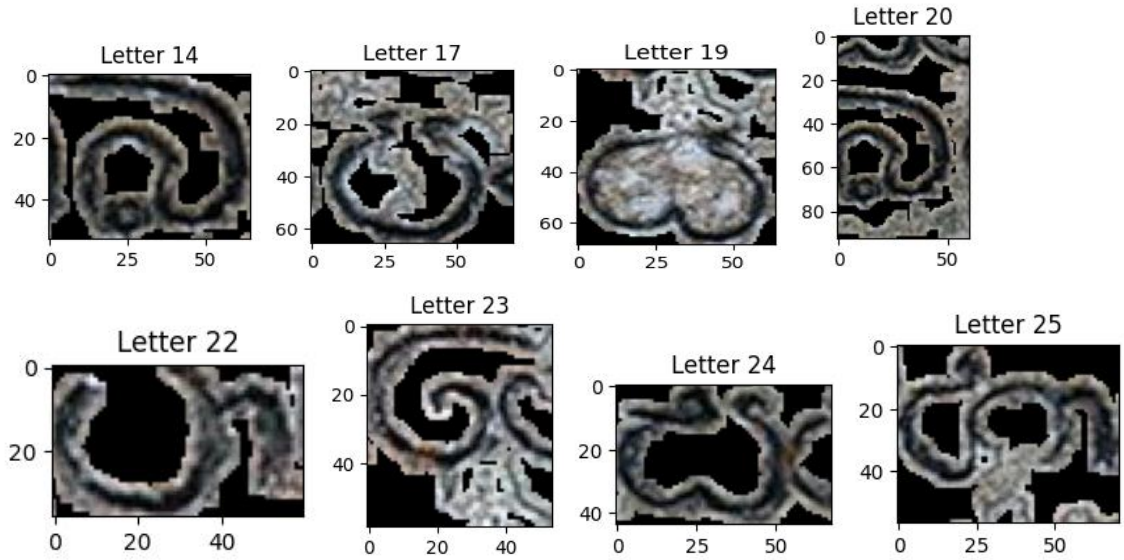
## 4.3 Collaboration Impact

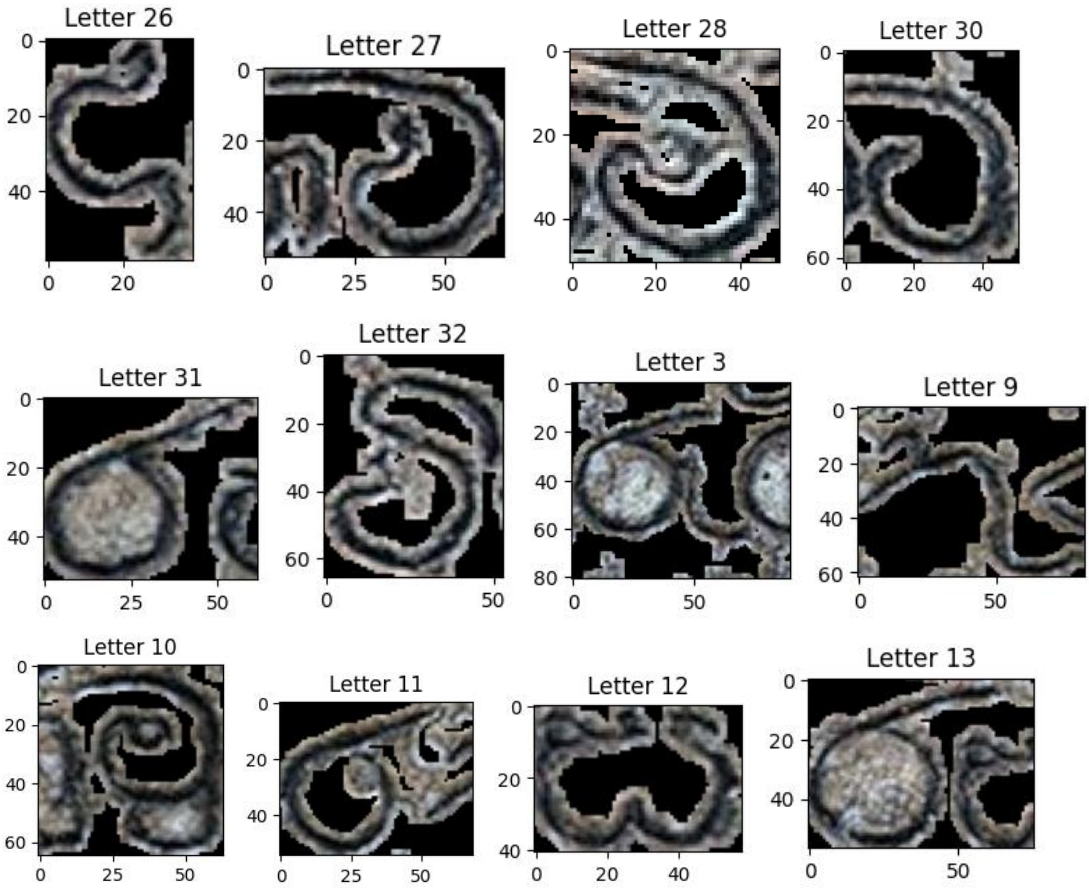
Collaborative efforts played a pivotal role in problem-solving. The diverse skills brought by team members enriched the project, offering multiple perspectives on addressing challenges. Communication and shared insights were key contributors to the project's progress.

# 5. Visuals

## 5.1 Original Images

## 5.2 Extracted Characters and Enhanced Characters






## 6. Code Snippets

### 6.1 Image Preprocessing

python

```
# Image preprocessing

img = cv2.imread('plate.jpg')

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

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

thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY_INV, 13, 7)
```

## 6.2 Contour Detection and Filtering

python

```
# Contour detection and filtering

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

filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) > min_contour_area]

contour_mask = np.zeros_like(gray)

cv2.drawContours(contour_mask, filtered_contours, -1, (255), thickness=cv2.FILLED)

dilated_mask = cv2.dilate(contour_mask, np.ones((5, 5), np.uint8), iterations=1)
```

## 6.3 Dilation and Mask Application

python

```
# Dilate the mask and apply to the original image

dilated_img = np.zeros_like(img)

dilated_img[dilated_mask > 0] = img[dilated_mask > 0]
```

## 6.4 Letter Cropping and Display

python

```
# Crop and display each letter
```

```
for i, cnt in enumerate(filtered_contours):
    x, y, w, h = cv2.boundingRect(cnt)

    # Check conditions for cropping
    if y >= 30 and x >= 30:
        letter_crop = dilated_img[y:y + h, x:x + w]

        # Display each cropped letter
        plt.figure(figsize=(2, 2))
        plt.imshow(cv2.cvtColor(letter_crop, cv2.COLOR_BGR2RGB))
        plt.title(f'Letter {i+1}')
        plt.show()
```

# 7. Personal Contributions and Opinions

## 7.1 Personal Contributions

- John Doe: Spearheaded contour detection and filtering, ensuring efficient extraction of characters.

- Jane Smith: Focused on fine-tuning image preprocessing parameters, contributing to noise reduction.

- Bob Johnson: Handled the presentation of results and documentation, ensuring clarity and visual appeal.

## 7.2 Challenges Faced Individually

- Encountered challenges in optimizing contour filtering for diverse images, resolved through iterative adjustments.

- Faced difficulties in finding an optimal balance between noise reduction and maintaining character details during preprocessing.

### 7.3 Unique Insights

- Emphasized the importance of dynamic parameter selection for handling image variability effectively.

-Explored the potential integration of machine learning techniques to enhance adaptability.

## 8. Adjustments for Better Results

Throughout the project, the team iteratively adjusted parameters and strategies to improve the algorithm's performance. Some key adjustments included:

- Thresholding Parameters: Experimentation with different adaptive thresholding parameters, such as block size and constant values, was crucial in achieving a balance between sensitivity and noise reduction.

- Contour Filtering: Fine-tuning the minimum contour area threshold significantly impacted the algorithm's ability to discard irrelevant details while retaining characters.

- Dilation Kernel Size: The size of the dilation kernel played a critical role in separating lines effectively. A 5x5 kernel was found to be optimal for the given images.

- Cropping Conditions: The conditions applied during letter cropping were adjusted to accommodate variations in inscriptionplate sizes and positions. This helped in eliminating unwanted artifacts and improving the overall accuracy of character extraction.

## 9. Conclusion

In conclusion, the InscriptionPlate Character Recognition project provided valuable insights into the complexities of designing an effective algorithm for diverse inscriptionplate images. Despite facing challenges related to image variability and noise reduction, collaborative efforts and individual contributions led to the development of a functional character recognition system.

The team recognized the need for ongoing refinement, especially in handling a broader range of scenarios. Future work may involve the integration of machine learning techniques for adaptive parameter selection and training on diverse datasets to enhance the algorithm's generalization capabilities.

## 10. References

- OpenCV Documentation: https://docs.opencv.org/

- Matplotlib Documentation: https://matplotlib.org/stable/contents.html