Assignment 02

Image Processing

Name: M.S.Saheer

Index: KIC-HNDCSAI-Y2-231-F-001

Module: Image Processing

Group Members: M.S.Saheer, M.F.A.M.Anas

Contents

[Introduction 2](#_Toc152539946)

[Materials Used 2](#_Toc152539947)

[Image Acquisition 3](#_Toc152539948)

[Image Sources 3](#_Toc152539949)

[Importance of Good Lighting and Minimal Distortion 3](#_Toc152539950)

[Image Samples Collected 3](#_Toc152539951)

[Image Preprocessing 4](#_Toc152539952)

[1. Resizing 4](#_Toc152539953)

[2. Denoising 4](#_Toc152539954)

[3. Histogram Equalization 4](#_Toc152539955)

[4. Varying Lighting Conditions 5](#_Toc152539956)

[Image Segmentation 5](#_Toc152539957)

[1. Cropping 5](#_Toc152539958)

[2. Thresholding 6](#_Toc152539959)

[3. Edge Detection 6](#_Toc152539960)

[Extraction 7](#_Toc152539961)

[Automatic Extraction 7](#_Toc152539962)

[Results 7](#_Toc152539963)

[Manual Extraction 8](#_Toc152539975)

[Results 10](#_Toc152539976)

[Challenges 11](#_Toc152539987)

[Insights Gained 11](#_Toc152539988)

[Conclusion 11](#_Toc152539989)

[References 12](#_Toc152539990)

# Introduction

The goal of this assignment is to develop and image processing program for the extraction characters from the Sellipi stone inscriptions found in various parts of Sri Lanka which was written long time back. Investigating and identifying characters could help us to understand more about our ancient culture and also may lead us to find solutions to future problems by learning how they solved those problems in ancient times which are written in Sellipi inscriptions.

The image extraction process involves image acquisition, pre-processing of the image, text segmentation, automatic character extraction, manual character extraction, character enhancement and report writing.

## Materials Used

1. Computer with Anaconda Environment & Jupiter Notebook installed.
2. OpenCV, Numpy and Matplotlib libraries.
3. High quality images of the Sellipi inscriptions.

# Image Acquisition

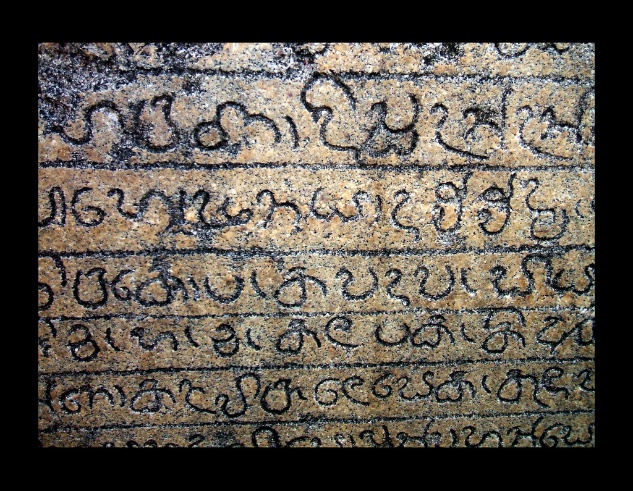
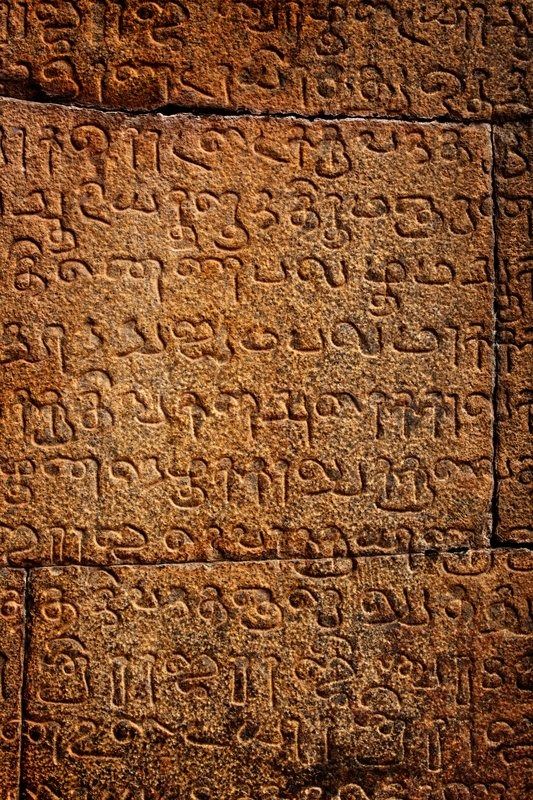
## Image Sources

The images used in the project were found through internet from various resources. The images used showcase different writings, styles, variations and distortions. Finding high quality image of Sellipi was a challenging task but by deep investigation we were able to collect some images for our project.

## Importance of Good Lighting and Minimal Distortion

To get the best results we prioritized good lighting conditions and minimum distortion during the image acquisition process. This ensures that the subsequent image processing steps are performed on high-quality data.

## Image Samples Collected



# Image Preprocessing

As the preprocessing techniques after applying different techniques, we identified the most effective ones for our images. Which are resizing, denoising and histogram equalization.

## Resizing

We standardized all the images in to common size for the consistency in the effects applied and to make the cropping process easier in the final step. By resizing we saved a lot of time as we didn’t need to adjust the size when testing for different images. The default size we went is 800 X 600.

width, height = 800, 600

resized\_image = cv2.resize(original\_image, (width, height), interpolation=cv2.INTER\_AREA)

## Denoising

For the denoising we have used two methods. One is median filtering and the other is Fast no-local means denoising. First Median is applied to blur the image making some small noises to disappear and create a flat picture for Fast no-local means denoising. Then Fast no-local means denoising was applied which resulted in reduction of noise which were created from the irregularity of the stone and it enhanced the image clarity for the next process.

median\_image = cv2.medianBlur(resized\_image, 5)

The parameter 5 specifies the size of the kernel, representing the neighborhood for calculating the median value.

denoised\_image = cv2.fastNlMeansDenoisingColored(median\_image, None, h=10, templateWindowSize=7, searchWindowSize=21)

h = 10 control the filter strength, templateWindowSize specifies the size of the pixel neighborhood, and searchWindowSize defines the size of the search neighborhood.

## Histogram Equalization

By processing through histogram equalization, we were able to improve the contrast of the denoised image, making small details of the characters to stand out. We experimented with normal histogram equalization and adaptive histogram equalization but for some images the normal one gave more prominent result and for other adaptive equalization was effective.

equalized\_image = cv2.equalizeHist(gray\_image)

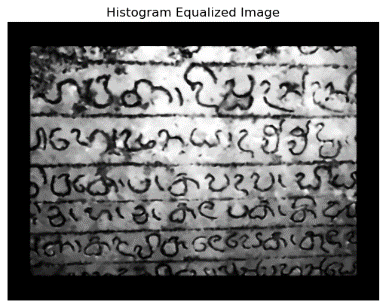
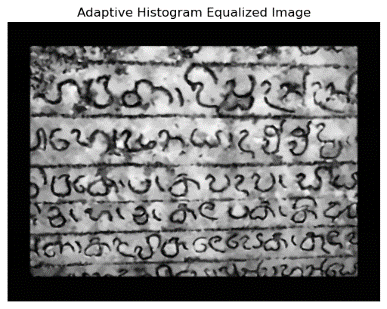
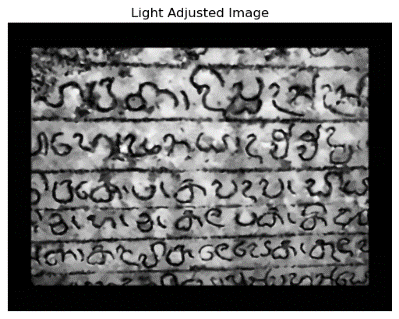
clahe = cv2.createCLAHE(clipLimit = 2.0, tileGridSize=(8, 8))

aequalized\_image = clahe.apply(gray\_image)

## Varying Lighting Conditions

Lighting conditions are varied to give a uniformity to the light spread in the image. Using gamma corrections contrast of the image is adjusted.

gamma = 1.2

adjusted\_image = np.clip((aequalized\_image / 255.0) \*\* gamma \* 255.0, 0,255).astype(np.uint8)

# Image Segmentation

To separate Sellipi characters from the image we have mainly used Thresholding and Canny Edge Detection techniques.

## Cropping

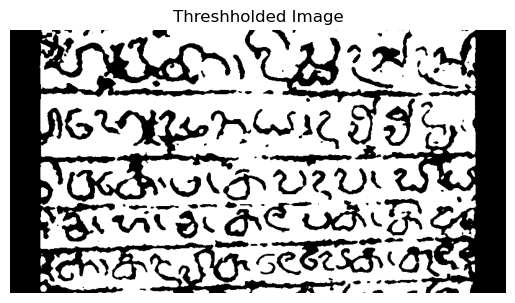


First, we cropped out the image to only focus on the region of interest where the Sellipi characters are located. So, all the noises and distortions from the surrounding elements are discarded.

x, y, w, h = 0, 100, 800, 425

cropped\_image = aequalized\_image[y:y+h, x:x+w]

## Thresholding

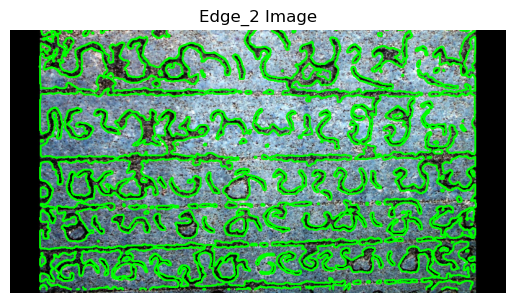
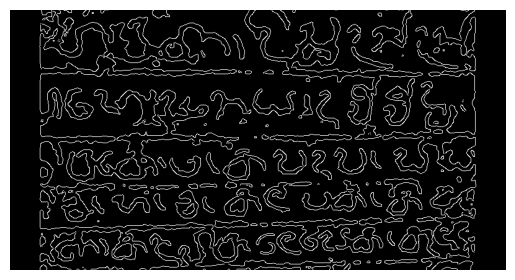


Adaptive thresholding is experimented and applied to achieve binary segmentation. Which resulted in a black and white image where all the values below our minimum threshold is set to white and above the maximum threshold is set black.

\_, thresholded\_image = cv2.threshold(cropped\_image, 50, 200, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

Here 50 and 200 represents the minimum and maximum values for the thresholding.

## Edge Detection

For the edge detection we have Canny Edge Detection from the opencv library. Canny edge detection resulted in precise boundary identification of Sellipi text.  

edges = cv2.Canny(thresholded\_image, 50, 150)

contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

segmented\_image = resized\_image[y:y+h, x:x+w].copy()

cv2.drawContours(segmented\_image, contours, -1, (0, 255, 0), 2)

# Extraction

For character extraction we experimented with two methods. One is manual extraction which allows the user to mask the Sellipi character and separate it from the image as a separate image. This separated image again will go through the all preprocessing methods mentioned in topic 2 and the resulting enhanced character will be saved in a separate folder. The other method is the automatic method wich will identify the patterns in the edges and crops the images and separate them.

## Automatic Extraction

for i, contour in enumerate(contours):

    x, y, w, h = cv2.boundingRect(contour)

    # Ensure the contour is not too small (adjust this threshold based on your needs)

    if w > 20 and h > 20:

        # Exclude contours with a specific aspect ratio (e.g., long lines)

        aspect\_ratio = w / h

        aspect\_ratio\_threshold = 5  # Adjust this threshold based on your needs

        if aspect\_ratio < aspect\_ratio\_threshold:

            # Crop the letter from the original image

            letter = thresholded\_image[y:y + h, x:x + w]

            # Save the cropped letter as a PNG file

            letter\_filename = os.path.join(output\_dir, f"letter\_{i}.png")

            cv2.imwrite(letter\_filename, letter)

This code snippet iterates via the contours identified in the previous step. For every contour, it retrieves the bounding rectangle using cv2.BoundingRect. To make sure the contour represents a valid character, it assessments if both width (w) and peak (h) are greater than 20 pixels. Additionally, it excludes contours with an thing ratio above a certain threshold (five in this situation) to dispose of long strains. For legitimate contours, the corresponding location within the thresholded photograph is cropped, and every cropped character is stored as a PNG document in the detailed output listing (output\_dir).

### Results

### 

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Manual Extraction

# Create a copy of the edge image for visualization purposes

edge\_visualization = cv2.cvtColor(edges, cv2.COLOR\_GRAY2BGR)

# Initialize empty list to store extracted characters

extracted\_characters = []

# Create a named window for interactive selection

cv2.namedWindow('Manual Character Extraction')

# Mouse callback function for interactive selection

def mouse\_callback(event, x, y, flags, param):

    global drawing, ix, iy, roi\_points

    if event == cv2.EVENT\_LBUTTONDOWN:

        drawing = True

        ix, iy = x, y

        roi\_points = [(x, y)]

    elif event == cv2.EVENT\_LBUTTONUP:

        drawing = False

        roi\_points.append((x, y))

        # Draw a rectangle around the selected region

        cv2.rectangle(edge\_visualization, (ix, iy), (x, y), (0, 255, 0), 2)

        cv2.imshow('Manual Character Extraction', edge\_visualization)

        # Convert coordinates to match the resized image

        ix\_resized, iy\_resized = int(ix \* (resized\_image.shape[1] / edges.shape[1])), int(iy \* (resized\_image.shape[0] / edges.shape[0]))

        x\_resized, y\_resized = int(x \* (resized\_image.shape[1] / edges.shape[1])), int(y \* (resized\_image.shape[0] / edges.shape[0]))

        # Extract the character within the selected region from the resized image

        extracted\_character = resized\_image[iy\_resized:y\_resized, ix\_resized:x\_resized]

        extracted\_characters.append(extracted\_character)

# Set up the mouse callback

cv2.setMouseCallback('Manual Character Extraction', mouse\_callback)

# Main loop for interactive selection

drawing, ix, iy, roi\_points = False, -1, -1, []

while True:

    cv2.imshow('Manual Character Extraction', edge\_visualization)

    key = cv2.waitKey(1) & 0xFF

    # Press 'ESC' to exit the interactive selection

    if key == 27:

        break

# Convert the list of extracted characters to a NumPy array

extracted\_characters = np.array(extracted\_characters)

# Save extracted characters for future use

for i, character in enumerate(extracted\_characters):

    cv2.imwrite(f'extracted\_character\_{i + 1}.png', character)

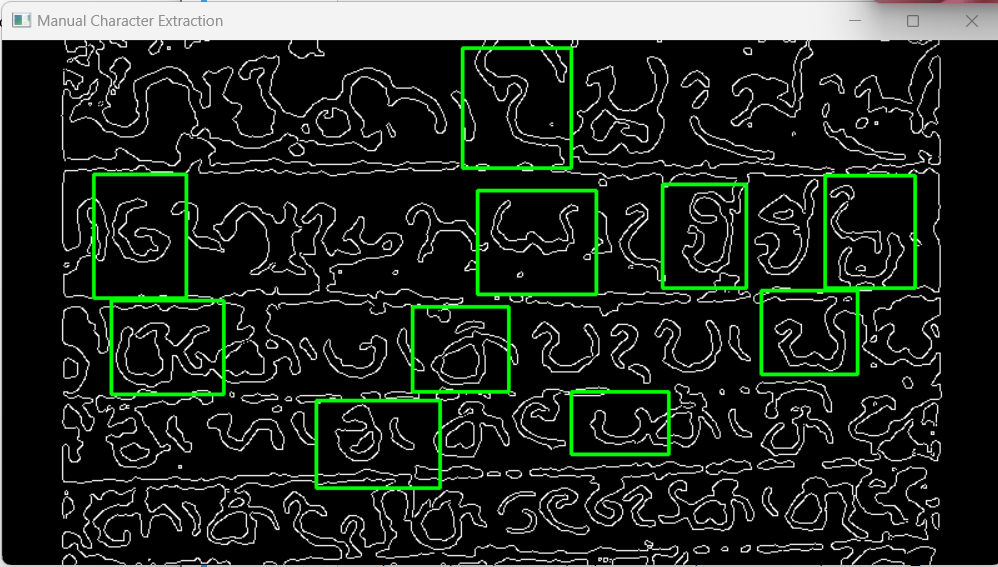
# Release resources

cv2.destroyAllWindows()

This code snippet implements an interactive person extraction device the usage of a mouse callback feature. The consumer can select areas of interest by using clicking and dragging the mouse. The decided on areas are visualized through drawing rectangles on the edge-detected image (edge\_visualization). The coordinates are adjusted to suit the resized picture dimensions.

Once the user finalizes the selection, the extracted characters are stored within the extracted\_characters list. These characters are then converted into a NumPy array and saved as person PNG documents. The filenames are based at the order of extraction.

The code permits for specific and manual extraction of characters from the resized photo, supplying a person-friendly approach for refining the segmentation process.



## Results

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Challenges

1. Variety of inscription styles  
   Addressing the diversity in Sellipi inscription styles posed a massive challenge during segmentation. The adaptability of segmentation techniques to distinct patterns required careful attention and quality-tuning.
2. Contrast and lighting condition changes  
   Achieving consistent comparison across pixels with varying lighting situations presented a routine project. The effectiveness of enhancement techniques, specially at some stage in pre-processing, had to be carefully balanced to accommodate those variations.
3. Stone irregularities  
   Most of the stones where carvings are written they have been decayed or some sort of black color is formed above it so detecting them was a very hard task we used some filtering techniques specially no-local means denoising filter to smooth out those irregularities.

# Insights Gained

1. Manual and Automatic Extraction  
   We have gained the knowledge on how to manually detect patterns using edge detection methods mainly with Canny edge detection and spent most time tuning this edge detection technique.
2. Image Segmentation  
   Learned some of the image segmentation techniques including thresholding and edge detection. Also had the chance to refer some articles related to using K-Means to cluster images / segment them.
3. Continuously iterating with different parameters  
   Image processing project needed continuous iterations and adjustments of parameters. This iteration process helped us in understanding how different filters impact on different levels of image processing and also able to understand how the parameter work for different processing methods.

# Conclusion

Overcoming the demanding situations and gaining insights from the image processing assignment not simplest resulted in an powerful answer for Sellipi individual extraction but additionally contributed valuable lessons for future initiatives. The collaborative efforts of the group and the iterative nature of the process underscored the significance of adaptability and user interaction in attaining accurate and significant consequences.

Project Files: https://drive.google.com/drive/folders/11dzw\_CScW\_77dryV\_hI\_VcBvd6HEEgBx?usp=sharing

# References

1. <https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.flickr.com%2Fphotos%2F47969185%40N05%2F5048247184&psig=AOvVaw3Rd7o_KSpI3IwLo2xR8ZyO&ust=1701713863653000&source=images&cd=vfe&ved=0CBIQjRxqFwoTCLiYrOLv84IDFQAAAAAdAAAAABAE>
2. <https://www.google.com/url?sa=i&url=https%3A%2F%2Fsinhala.alpanthiya.lk%2Fsellipi-wala-sandhahan-purana-sri-lankawe-parani-adhikarana-kramaya%2F&psig=AOvVaw1Ys4geixxL3E4A3dmgWr3S&ust=1701713876899000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCLjIiujv84IDFQAAAAAdAAAAABAP>
3. <https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html>
4. <https://www.researchgate.net/publication/332947534_Image_Based_Hieroglyphic_Character_Recognition>