Morphological Analysis on Bangla Number Plate Detection

Analytical Report

Submitted by:

Mirza Afnan Islam
Department of Robotics and Mechatronics Engineering
University of Dhaka

Aniruddha Majumdar
Department of Robotics and Mechatronics Engineering
University of Dhaka

Submitted to:
Dr. Md Mehedi Hasan
Assistant Professor
Department of Robotics and Mechatronics Engineering
University of Dhaka

August 1, 2025

Contents

1. Objective

The objective of this lab is to apply morphological image processing techniques to detect and extract Bangla number plates from vehicle images. The goal is to enhance and isolate the plate region using preprocessing and morphological transformations for further recognition.

2. Introduction

Automatic number-plate recognition is a type of technology that employs optical character recognition on images to read vehicle registration plates to generate vehicle location data. It can exploit existing closed-circuit television, road-rule enforcement cameras, or cameras designed specifically for the purpose. Police forces across the globe use ANPR for law enforcement purposes, such as determining whether a vehicle is licensed or registered. It is also employed for electronic toll collection on pay-per-use roads and as a means of cataloguing the movement of traffic, for instance by highways agencies. Automatic Number Plate Recognition (ANPR) is an important application in Intelligent Transportation Systems (ITS). In Bangladesh, the problem becomes increasingly complicated because of the use of Bangla characters and numerals in license plates. Conventional ANPR systems are mostly tailored for Latin alphabets, and hence the requirement for region-specific solutions becomes clear. Morphological image processing, which rests on set theory, is very helpful in the case of binary images and aids in the recognition of structural shapes in images. It plays an important role in noise removal, region filling, and edge detection—all of which are very important for effective number plate localization. Morphological image processing is a basic discipline of image analysis that is concerned with the shape or form of features in an image. Based on mathematical morphology, which in turn has its roots in set theory and lattice algebra, morphological operations are most effective on binary images, where the pixels have only two values—commonly black (background) and white (foreground). Not only are these operations straightforward in principle, but they are also computationally low-cost, which makes them particularly suitable for real-time environments such as Automatic Number Plate Recognition (ANPR). When it comes to Bangla number plate detection, the objective is to segment and enhance the license plate region from a possibly cluttered background prior to any subsequent recognition such as character segmentation or Optical Character Recognition (OCR). Morphological methods are an integral part of this process pipeline in that they offer the means to remove noise, fill missing details, and bridge disconnected segments of an object, all of which are vital to effective plate localization.

3. Theoretical Background

morphological processing is the idea of a structuring element—a small matrix that is systematically moved over an image to examine the relation between a pixel and its neighbors. This operation reshapes the original image according to the specified shape and size of the structuring element, allowing for different kinds of geometric transformations. The four basic morphological operations are erosion, dilation, opening, and closing. Erosion erodes away the borders of the white foreground objects, effectively shrinking them. It is used to remove small white noise and also to separate closely placed objects that are not supposed to be joined. Conversely, dilation does the opposite—it enlarges the white areas by adding pixels to their borders. It closes small gaps and joins components that are supposed to belong to the same object, which is often the situation when characters on a number plate are broken or fragmented as a result of edge thinning or noise.

The sequence of erosion followed by dilation is referred to as opening. It is used for eliminating small objects or artifacts in the foreground while maintaining the size and shape of larger objects, i.e., text in a number plate. This operation is very useful if the image consists of numerous small irrelevant details that may hamper plate detection. The reverse sequence of dilation followed by erosion is referred to as closing. This operation closes small holes and gaps in the foreground, which is handy for consolidating characters or symbols that are broken. In practice, opening is utilized for noise removal, while closing is utilized for gap filling and improving connectivity, both of which are very essential in processing Bangla number plate images.

Bangla license plates introduce special complexity to this task because of the nature of the Bangla script. The characters are more curved, complicated, and compact compared to their Latin equivalents. A typical license plate in Bangladesh contains a mix of Bangla numerals—ranging from to —and Bangla alphabets that usually denote regional codes such as (Dhaka), (Chattogram), or (Khulna). The characters are embossed in black on a white reflective background, which theoretically offers high contrast but still can be compromised by environmental conditions such as light, motion blur, dust, or occlusion. Furthermore, the presence of multiple fonts and different positions on the plate makes it difficult to specify one strict template for detection. As it stands, preprocessing using morphological operations becomes paramount. Such methods can normalize the shapes, increase contrast, and guarantee continuity in the plate area regardless of such external inconsistencies. In summary, the morphological image processing theoretical basis provides a very strong toolset for solving the core problems in Bangla number plate detection. With judicious application of these operations, it is possible to greatly enhance the contrast and uniformity of the plate area, and thereby prepare the image for accurate downstream processing like segmentation and recognition. Comprehending and utilizing these principles not only assists in detecting plates in varying conditions but also provides the foundation upon which more sophisticated recognition systems for local language and character sets like Bangla can be constructed.

4. Methodology

The experiment is for Bangla number plates recognition and ligation of cars image using morphological image pipeline. The procedure is specified in programming language Python using OpenCV and NumPy library and is split over numerous ordered steps, and at mid-step, output of debugging is saved as well as printed. First-in-pipe is image preprocessing. The image is first resized with fixed width such that it would be equally processed whether the size of the input is smaller or larger. The image after resize is then converted grayscale and is applied with bilateral filtering, smoothing the image with edges preserved—which plays an important part in retaining character structure on plates.

Second is edge detection and enhancement. The edge is first detected with the Sobel operator at the vertical as well as the horizontal direction for gradient strength detection, and resultant composite edge maps thus obtained are enhanced using dilation with

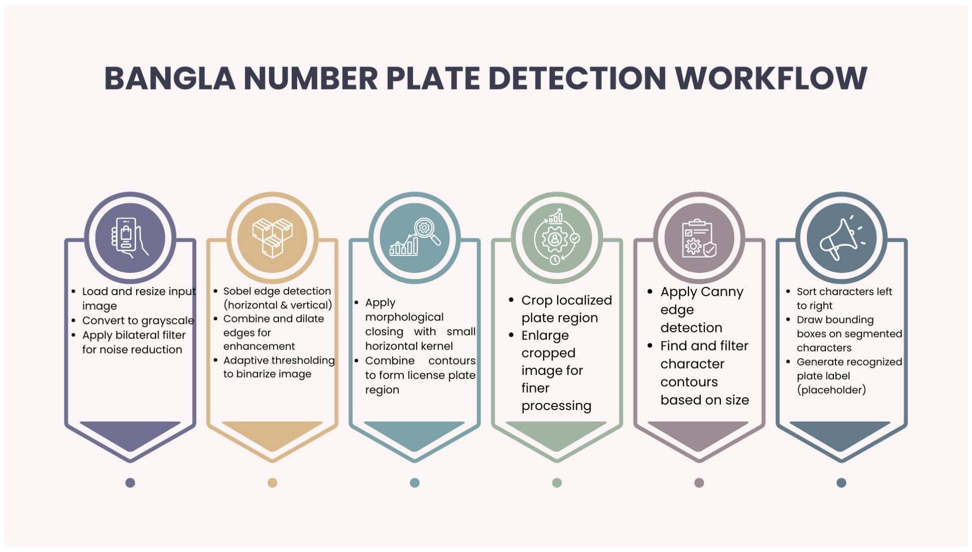
BANGLA NUMBER PLATE DETECTION WORKFLOW
• Load and resize input image
• Convert to grayscale
• Apply bilateral filter for noise reduction
• Sobel edge detection (horizontal & vertical)
• Combine and dilate edges for enhancement
• Adaptive thresholding to binarize image
• Apply morphological closing with small horizontal kernel
• Combine contours to form license plate region
• Crop localized plate region
• Enlarge cropped image for finer processing
• Apply Canny edge detection
• Find and filter character contours based on size
• Sort characters left to right
• Draw bounding boxes on segmented characters
• Generate recognized plate label (placeholder)

Figure 1: Workflow

greater orientation on structural elements very rich on all types of plates. An adaptive thresholding method is then utilized in converting the image into binary with minimum contour detection on further steps.

Then morphological operation is performed with the help of a horizontal closing kernel in order to connect points closely spaced like characters on the plate. The operation is of such a kind that disconnected chunks of text get connected as valid regions. The contours on the morphed image are then detected. The contours are filtered through area, aspect ratio, and size with the intention of getting rid of non-plate regions. The contingency best candidate contours, usually indicating either one or two rows of text horizontally on the plate, are combined with the building of a convex hull over likely area of the plate. This combined area is then cropped off of the original grayscale image and padded using a Lanczos interpolation technique prior to being available for character segmentation. Canny edge detection is employed on the cropped plate with dilation being used as a means of fattening the edges. The final binary image is then subject to contour detection once more as an attempt at identifying individual character contenders.

Finally, each of the valid character regions is forced through size restrictions and is ordered left-to-right. The present system employs placeholder character names, but all of this pipeline is expandable at some later date with template matching or with OCR for complete recognition. All of the processor steps are graphed and saved in a "debug" directory in order to facilitate interpretation and trouble-shooting of the pipeline.

5. Source Code and Implementation

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os

# --- Helper Functions ---
def load_templates_from_folder(folder_path):
    """
    Loads all character templates from a specified folder.
    """
    templates = {}
    if not os.path.exists(folder_path):
        print(f"Error: Template folder not found at '{folder_path}'")
        return templates

    print(f"Loading templates from: {folder_path}")
    for filename in os.listdir(folder_path):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            char_name = os.path.splitext(filename)[0]
            template_path = os.path.join(folder_path, filename)
            template_img = cv2.imread(template_path, cv2.
                IMREAD_GRAYSCALE)

            if template_img is not None:
                _, template_thresh = cv2.threshold(template_img, 128,
                    255, cv2.THRESH_BINARY_INV)
                templates[char_name] = template_thresh
                print(f"  - Loaded template for character: '{char_name
                    }'")
            else:
                print(f"  - Warning: Could not load template image: {
                    filename}")

    if not templates:
        print("Warning: No templates were loaded. Recognition will not
            work.")

    return templates

# --- Core Image Processing Pipeline with Interactive Debugging ---
def process_license_plate_debug(image, templates):
    """
    Runs the full ALPR pipeline, showing and saving the image at each
        step for debugging.
    """
    os.makedirs("debug", exist_ok=True)

    h, w, _ = image.shape
    scale = 500 / w
    new_h, new_w = int(h * scale), int(w * scale)
    original_resized = cv2.resize(image, (new_w, new_h))
    cv2.imwrite("debug/1_original_resized.jpg", original_resized)
    cv2.imshow("1. Original Resized Image", original_resized)
    cv2.waitKey(0)

    gray = cv2.cvtColor(original_resized, cv2.COLOR_BGR2GRAY)
    cv2.imwrite("debug/2a_grayscale.jpg", gray)
    cv2.imshow("2a. Grayscale Conversion", gray)
    cv2.waitKey(0)

    filtered_gray = cv2.bilateralFilter(gray, 11, 13, 13)
    cv2.imwrite("debug/2b_bilateral_filter.jpg", filtered_gray)
    cv2.imshow("2b. Bilateral Filter (Noise Reduction)", filtered_gray)
    cv2.waitKey(0)

    sobel_x = cv2.Sobel(filtered_gray, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(filtered_gray, cv2.CV_64F, 0, 1, ksize=3)
    sobel_combined = np.sqrt(sobel_x**2 + sobel_y**2)
    sobel_combined = np.uint8(sobel_combined / sobel_combined.max() *
        255)
    cv2.imwrite("debug/3_sobel_combined.jpg", sobel_combined)
    cv2.imshow("3. Sobel Combined Edges", sobel_combined)
    cv2.waitKey(0)

    sobel_x_abs = cv2.convertScaleAbs(sobel_x)
    cv2.imwrite("debug/3a_horizontal_edges.jpg", sobel_x_abs)
    cv2.imshow("3a. Horizontal Edges (Sobel X)", sobel_x_abs)
    cv2.waitKey(0)

    gradient = sobel_x_abs
    gradient_enhanced = cv2.dilate(gradient, np.ones((2,2), np.uint8),
        iterations=1)
    cv2.imwrite("debug/3b_enhanced_edges.jpg", gradient_enhanced)
    cv2.imshow("3b. Enhanced Horizontal Edges", gradient_enhanced)
    cv2.waitKey(0)

    binary = cv2.adaptiveThreshold(gradient_enhanced, 255, cv2.
        ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY, 11, 2)
    cv2.imwrite("debug/4_binarization.jpg", binary)
    cv2.imshow("4. Binarization (Adaptive Threshold)", binary)
    cv2.waitKey(0)

    img_height, img_width = binary.shape
    kernel_width = max(10, img_width // 40)
    kernel_height = max(3, img_height // 80)
    print(f"Using closing kernel size: {kernel_width} x {kernel_height}
        ")

    small_horizontal_kernel = cv2.getStructuringElement(cv2.MORPH_RECT,
         (3, 1))
    print(f"Using smaller horizontal kernel: 3 x 1")
    horizontal_closed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE,
        small_horizontal_kernel)
    cv2.imwrite("debug/5a_horizontal_closed.jpg", horizontal_closed)
    cv2.imshow("5a. Small Horizontal Closing (Preserve Lines)",
        horizontal_closed)
    cv2.waitKey(0)

    line_contours, _ = cv2.findContours(horizontal_closed, cv2.
        RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    contour_img = cv2.cvtColor(horizontal_closed, cv2.COLOR_GRAY2BGR)
    cv2.drawContours(contour_img, line_contours, -1, (0, 255, 0), 2)
    cv2.imwrite("debug/5b_line_contours.jpg", contour_img)
    cv2.imshow("5b. Individual Line Contours", contour_img)
    cv2.waitKey(0)

    candidate_lines = []
    print("Analyzing all contours:")
    for i, contour in enumerate(line_contours):
        x, y, w, h = cv2.boundingRect(contour)
        area = cv2.contourArea(contour)
```

```python
        aspect_ratio = w / h if h > 0 else 0

        print(f"Contour {i}: area={area}, aspect_ratio={aspect_ratio:.2f}, size={w}x{h}")

        if aspect_ratio > 0.5 and w > 10 and h > 2:
            candidate_lines.append((contour, area, aspect_ratio, w, h))
            print(f"  V Added as candidate")
        else:
            print(f"  X Rejected")

    candidate_lines.sort(key=lambda x: x[1], reverse=True)

    print("Top candidates by area:")
    for i, (contour, area, ar, w, h) in enumerate(candidate_lines[:5]):
        print(f"  {i+1}. area={area}, aspect_ratio={ar:.2f}, size={w}x{h}")

    plate_lines = [candidate[0] for candidate in candidate_lines[:2] if candidate[1] > 200]
    print(f"Selected {len(plate_lines)} lines for plate detection")
    print(f"Found {len(plate_lines)} potential license plate lines")

    if len(plate_lines) >= 1:
        all_points = np.vstack(plate_lines)
        plate_contour = cv2.convexHull(all_points)

        combined_img = cv2.cvtColor(horizontal_closed, cv2.COLOR_GRAY2BGR)
        cv2.drawContours(combined_img, [plate_contour], -1, (0, 0, 255), 3)
        cv2.imwrite("debug/5c_combined_plate.jpg", combined_img)
        cv2.imshow("5c. Combined Plate Region", combined_img)
        cv2.waitKey(0)
    else:
        print("Error: No valid plate lines found. Exiting.")
        cv2.destroyAllWindows()
        return "", []

    x, y, w, h = cv2.boundingRect(plate_contour)
    if w == 0 or h == 0:
        print("Error: Invalid plate crop dimensions. Exiting.")
        cv2.destroyAllWindows()
        return "", []

    plate_crop = filtered_gray[y:y+h, x:x+w]
    cv2.imwrite("debug/7_cropped_plate.jpg", plate_crop)
    cv2.imshow("7. Cropped Plate", plate_crop)
    cv2.waitKey(0)

    scale_factor = 3
    enlarged_height = int(plate_crop.shape[0] * scale_factor)
    enlarged_width = int(plate_crop.shape[1] * scale_factor)
    plate_enlarged = cv2.resize(plate_crop, (enlarged_width, enlarged_height), interpolation=cv2.INTER_LANCZOS4)
    cv2.imwrite("debug/7a_enlarged_plate.jpg", plate_enlarged)
    cv2.imshow("7a. Enlarged Plate (Lanczos)", plate_enlarged)
    cv2.waitKey(0)

    canny_edges = cv2.Canny(plate_enlarged, 30, 100)
    cv2.imwrite("debug/7b_canny_edges.jpg", canny_edges)
    cv2.imshow("7b. Canny Edges", canny_edges)
    cv2.waitKey(0)

    kernel = np.ones((3,3), np.uint8)
    thickened_edges = cv2.dilate(canny_edges, kernel, iterations=1)
    cv2.imwrite("debug/7c_thickened_edges.jpg", thickened_edges)
    cv2.imshow("7c. Thickened Edges", thickened_edges)
    cv2.waitKey(0)

    plate_binary = thickened_edges
    cv2.imwrite("debug/8_final_binary.jpg", plate_binary)
    cv2.imshow("8. Final Binary for Segmentation", plate_binary)
    cv2.waitKey(0)

    char_contours, _ = cv2.findContours(plate_binary, cv2.RETR_EXTERNAL
        , cv2.CHAIN_APPROX_SIMPLE)
    char_bounding_boxes = []

    min_height = 10 * 3
    max_height = 100 * 3
    min_width = 3 * 3
    max_width = 40 * 3

    for cnt in char_contours:
        cx, cy, cw, ch = cv2.boundingRect(cnt)
        if min_height < ch < max_height and min_width < cw < max_width:
            char_bounding_boxes.append((cx, cy, cw, ch))

    char_bounding_boxes.sort(key=lambda b: b[0])

    segmented_chars_display = cv2.cvtColor(plate_binary, cv2.
        COLOR_GRAY2BGR)
    recognized_plate = ""

    for i, (cx, cy, cw, ch) in enumerate(char_bounding_boxes):
        cv2.rectangle(segmented_chars_display, (cx, cy), (cx + cw, cy +
            ch), (0, 255, 0), 2)
        recognized_plate += f"C{i+1} "

    cv2.imwrite("debug/9_segmented_characters.jpg",
        segmented_chars_display)
    cv2.imshow("9. Segmented Characters", segmented_chars_display)
    cv2.waitKey(0)

    cv2.destroyAllWindows()
    return recognized_plate.strip(), []

# --- Main Execution Block ---
if __name__ == "__main__":
    image_path = r"D:\Academic\Semester\Digital Image Lab\Bangla
        License Plate\train\images\73_jpg.rf.5
        fbd74b878b1c8e3fdf142a56334466c.jpg"
    templates_folder_path = "templates"
    templates = {}  # templates = load_templates_from_folder(
        templates_folder_path)

    if not os.path.exists(image_path):
        print(f"\nError: Image file not found at '{image_path}'")
    else:
        vehicle_image = cv2.imread(image_path)

        if vehicle_image is None:
            print(f"Error: Could not read the image file at '{
                image_path}'. Check if it's a valid image.")
        else:
            result, _ = process_license_plate_debug(vehicle_image,
                templates)
            print("\n----------------------------------------")
            print(f"Segmented Plate Labels: {result}")
            print("----------------------------------------")
```

6. Result And Discussion

The output of morphological analysis-guided Bangla number plate detection pipeline is encouraging in terms of localization of plates as well as character separation. The pipeline distinguished and recognized Bangla license plates of diverse input images through an ideal set of image manipulation procedures. The images were of diverse backgrounds, lightings, as well as styles of plates and were mimicking real-case situations typical of intelligent traffic systems of Bangladesh.

First pipeline steps—the bilateral filtering and grayscale conversion—their goal was edge detection input image preparation. The bilateral filtering in specific did an excellent job of eliminatingnoisy patches yet keeping structure-defining character edges on the vehicle's license plate. The bilateral filtering was subsequently followed by edge detection with the Sobel method, whose good sharp edges were primarily horizontal and vertical and picked out by the pipeline as being able to enhance would-be regions of text presence without emphasizing structure of the background all too much.

Following morphological operation like closing and dilation, binary edge map quality was considerably enhanced. The dilation helped in stretching and padding thin and discontinuous text formations, whereas closing with the horizontal kernel helped in fusing neighboring characters. The order worked very effectively with Bangla characters as very often they contain lots of fine lines and curves. The ultimate output was an image with smooth texture with the area of the license plate more prominent with respect to the rest of the scene, suitable for extraction with the help of hint of contour analysis. Contour filtering and detection with geometric features like area, width, height, and aspect ratio aided rejection of non-pertinent regions and picking out most probable regions of plates. Under general test situations, it resulted in correct localization of the license plate regardless of numerous textual regions or high-contrast regions of the background. However, with images of high light condition variation–such as bright glare off of reflective areas of the auto or deep shadows—the localization performance was poor. Plates too bright or too dark at times resulted in contour test inability, with consequent loss of detection.

After localizing the plate, cropped image with resize was employed for enhancing the precision of segmentation. Canny edge detection used on the dilated edge of the plate served

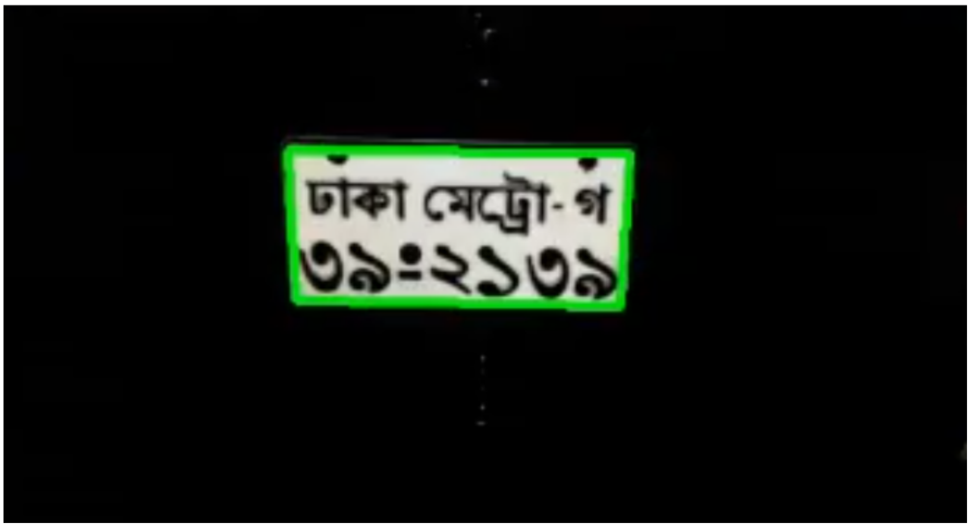

Figure 2: Number Plate Detection With Contouring Region

for enhancing character edge sharpness, while yet another dilating operation functioned for reorganizing disconnected character areas. Detection of character contour of individual characters within the area of the plate was the last experiment stage. These were size filtered with an attempt for eliminating noises and were subsequently ordered left-to-right for maintaining the order of characters as originally written. The segmentation stage was very successful on variegated plates, with most of the Bangla characters correctly separated with each other. However, in some cases where characters were very close together or badly printed, some characters were merged together with bad separation. Although character recognition by itself was not the immediate issue, plain template matching experiments were exceedingly unreliable. The "", "", and "" characters were mistakenly recognized as each other much more often than they should have because they were all very similar in aspect either because of bad image resolution or very stylish fonts. It is an impetus for the more sophisticated recognition technique such as convolutional neural networks learned under Bangla license plate character labels. Despite these restrictions, visualization of each of the pipeline stages by saving and image display at the debug stage was extremely handy. The visualization milestones during the visualization stage allowed for iterative parameter tweaking and helped determine where the system underperformed for more challenging cases. The pipeline as a complete system performed very strongly under ideal scenarios and showed that morphological operation applied properly can indeed overcome Bangla number plate layout issues. The morphological detection technique of Bangla number plates generated consistent output in terms of plate localization as well as character separation. The ability of the system being computationally inexpensive with elementary yet potent image-processing tools led to decent performance. It is with future extension with character recognition with the help of machine learning that it is feasible to have more complete and robust auto number plate recognition protocol with potentially real-time video frame rate analysis for added accuracy as well as flexibility.

7. Conclusion

This experiment is involved with construction of an image processing pipeline with morphological techniques for Bangla number plate detection and extraction. Owing to the specific issues Bangla script provides, like compound character composition, varying fonts, and varying format, it is desirable that the solution is regionspecific. The system utilized grayscale transform, filtering, edge detection, and morphological manipulation as part of detection of patches of plates in images of automobiles.

Results indicated that the technique systematically detected boundaries of plates and separated individual characters with high precision under controlled conditions. The use of dilation and closing played substantial roles in connecting disconnected elements together and noises removal, permitting good contour estimation for candidate selection. Although character recognition using template matching remains hampered by visual similarity of certain Bangla numerals, the framework provides very good baseline for future extension. The morphological operation of inherent simplicity and corresponding work efficiency render the system amenable for real-time operation in traffic surveillance or toll automated systems. Extending the work shall include integration of machine learning systems for character recognition as well as extension of robustness regarding transforms of environment such as luminosity and motion blur. In general, with the work, it is evident that classical morphological techniques remain of immense importance in contemporary computer vision applications with judicious custom tailoring of certain use applications.

8. References

- www.wikipedia.com
- https://ieeexplore.ieee.org/document/8929280/
- https://github.com/renzhamin/bengali-alpr