**Accredited SINTA 2 Ranking**
Decree of the Director General of Higher Education, Research, and Technology, No. 158/E/KPT/2021
Validity period from Volume 5 Number 2 of 2021 to Volume 10 Number 1 of 2026

Published online at: **http://jurnal.iaii.or.id**

**JURNAL RESTI**
**(Rekayasa Sistem dan Teknologi Informasi)**
Vol. 8 No. 6 (2024) 719 - 729 e-ISSN: 2580-0760

# The Memory Efficiency in a Receptionist Robot's Face Recognition System Using LBPH Algorithm

Endang Darmawan Yudi[1], Yesi Novaria Kunang[2]*, Ahmad Zarkasi[3]
[1], Master of Informatics Engineering, Universitas Bina Darma, Palembang, Indonesia
[2]Intelligent Systems Research Group, Faculty of Sains Technology, Universitas Bina Darma, Palembang Indonesia
[3]Department of Computer Engineering, Faculty of Computer Science, Universitas Sriwijaya, Palembang, Indonesia
[1]endangdarmawanyudi123@gmail.com, [2]yesinovariakunang@binadarma.ac.id, [3]ahmadzarkasi@unsri.ac.id

***Abstract***

*This research aims to develop a memory-efficient face recognition system for a receptionist robot using the Local Binary Patterns Histogram (LBPH) algorithm. Given the computational limitations of the Raspberry Pi, the system utilizes optimization techniques including grayscale conversion, noise reduction, and contrast adjustment to enhance processing efficiency. Testing demonstrates that the face recognition accuracy achieves 80.5% to 85.5% in offline mode, and 72% to 81% in real-time mode, with variations due to lighting conditions and facial expressions. The robot's servo motors exhibit a response time between 1.945 and 3.561 seconds, enabling responsive and interactive user engagement. The results suggest practical benefits for deploying face recognition in resource-constrained environments, enhancing the efficiency of robotic receptionist applications.*

*Keywords: Face recognition; LBPH; memory efficiency; Raspberry Pi*

*How to Cite:* E. D. Yudi, Yesi Novaria Kunang, and A. Zarkasi, "The The Memory Efficiency in a Receptionist Robot's Face Recognition System Using LBPH Algorithm", *J. RESTI (Rekayasa Sist. Teknol. Inf.)* , vol. 8, no. 6, pp. 719 - 729, Dec. 2024.
*DOI*: https://doi.org/10.29207/resti.v8i6.6048

## 1. Introduction

Face recognition is one of the key components in the development of artificial intelligence [1], [2]. In everyday life, humans naturally recognize others' faces simply by observing their eyes and facial features. This information is then stored in the brain, allowing us to identify individuals in future encounters. In the field of technology, the concept of face recognition has been widely adopted for various purposes, such as communication, identity verification, and attendance systems [3], [4].

Facial recognition is part of a broader field known as computer vision, which is a technology that enables computers to see, detect, and process images like human vision [5]. Computer vision forms the foundation for many automation applications requiring visual analysis, including facial recognition. In this context, facial recognition works by identifying unique features on a person's face, such as the eyes, nose, and mouth, which are then processed and stored in a database [6], [7], [8]. When the system scans a face, this data is compared to the existing database to identify and verify the individual's identity.

Various techniques in computer vision have been developed for facial recognition, such as Viola-Jones, Eigenfaces, Fisherface, and Local Binary Patterns Histograms (LBPH)[9]. Each technique has its advantages and disadvantages, as well as different applications depending on specific needs [10]. Compared to other computer vision techniques, such as Eigenfaces and Fisherface, the LBPH algorithm demonstrates superior performance in environments with limited computational resources, such as Raspberry Pi. Unlike Eigenfaces, which are sensitive to variations in lighting, or Fisherfaces, which require more extensive training data, LBPH is more robust in handling diverse lighting conditions and requires less memory [9], [11], [12]. This makes LBPH an ideal choice for real-time face recognition in constrained environments where computational efficiency is a priority(1). This method uses local binary patterns to analyze the texture of an image, where pixel intensity

Received: 15-09-2024 | Accepted: 11-11-2024 | Published Online: 25-12-2024

values are compared with their neighbors to generate a binary pattern. LBPH then divides the image into several blocks, calculates the binary pattern histogram for each block, and uses this information to recognize the face.

The development of technology in this research encompasses an in-depth study of facial recognition technology, the use of Raspberry Pi 3 in system development [11], [13] and the latest innovations in the implementation of receptionist robots. The novelty of this research lies in the integration of these three aspects, with an emphasis on memory efficiency on the Raspberry Pi 3. Although various related studies exist, such as the work by Hossain on Local Binary Patterns Histogram (LBPH) [14], which demonstrates that this algorithm can provide good results in facial recognition by utilizing texture information, this research introduces a new concept to enhance memory management efficiency on the Raspberry Pi 3 in the context of facial recognition for receptionist robots.

LBPH was chosen in this study due to its ability to utilize smaller data sources compared to other algorithms, such as Eigenfaces or Fisherfaces, making it more suitable for resource-constrained devices like the Raspberry Pi. The Raspberry Pi, which has limitations in computational power and memory capacity, requires an algorithm that is efficient in data processing, and LBPH meets this need. While LBPH may be less robust to changes in viewpoint or pose, its advantages in data efficiency and performance make it superior for real-time applications on platforms with limited resources.

In this research, we will implement facial recognition technology using the Raspberry Pi as the main processing unit. Due to the resource constraints of the Raspberry Pi, efficient programming methods and approaches are necessary in terms of memory usage and overall efficiency. The system will use features such as Haar to detect facial features like eyes and mouth before applying the LBPH algorithm for complete facial recognition [15], [16]. By utilizing an efficient data algorithm, the output of this system will be applied to the receptionist robot, which is capable of greeting guests or providing necessary information, making it more interactive and efficient in carrying out receptionist tasks.

## 2. Research Methods

The overall flow of the LBPH algorithm-based facial recognition robot can be seen in Figure 1. The system begins with the process of face detection through a connected camera. Once a face is detected, the system automatically runs the LBPH (Local Binary Patterns Histogram) Face Recognition algorithm to identify whether the face is already in the database or dataset that stores data of previous guests.

If the LBPH algorithm finds that the face is already registered in the dataset, the next step is to display the corresponding guest's name. The name is retrieved from the dataset based on facial identification. After the guest's name is displayed, the system will proceed by asking if the guest would like to perform a check-out process. Check-out indicates that the guest has completed their visit and wishes to remove their data from the system. If the guest opts for check-out, the system will delete all data related to the guest's face from the dataset, ensuring that no personal information is stored longer than necessary. Once the data deletion is complete, the system ends the process. If the registered guest does not choose to check out, the system will make no changes to the dataset and will return to waiting for the next face detection, repeating the cycle.

On the other hand, if the detected face is not found in the dataset, the system recognizes that the guest is not yet registered. At this point, the system will offer the guest the option to check-in. Check-in is the process of registering a new guest into the system. If the guest chooses to check in, the first step is to ask the guest to enter their name. This name is required to associate the identity with the facial data that will be stored.

The decision to capture 50 images of each individual is based on the need to enhance the variability of the facial data. By taking multiple images from different angles and with slight variations in lighting and expressions, the system builds a more comprehensive dataset for each person. This approach helps improve the recognition accuracy and ensures that the model is robust enough to handle minor changes in the face's appearance during real-time recognition.

After the guest's name is entered, the system will then capture 50 images of the guest's face. The image capture is done gradually to ensure that the system has enough image variation to improve accuracy in recognizing the guest's face in the future. Each captured image will be stored in a special folder named after the newly entered guest. While this study did not implement data augmentation techniques, such as flipping, rotation, or adding noise to the images, using these methods could potentially increase the variability of the dataset. Augmented data can help the system generalize better by simulating different real-world conditions, thus enhancing the robustness of face recognition even further.

Once the 50 facial images are successfully captured and saved, the guest has successfully checked in. Their facial data is now in the dataset and will be used for verification in future instances. After the check-in process is complete, the system returns to its initial state, ready to detect the next face. However, if the guest whose face is not in the dataset chooses not to check in, the system will end the process and return to the initial stage, waiting for the next face detection without storing any data.

Overall, this system leverages the LBPH algorithm for facial recognition with a high level of accuracy and

 

efficiently manages guest data through automated check-in and check-out processes. In this way, the system can handle both new and registered guests in an easy, secure, and structured manner.

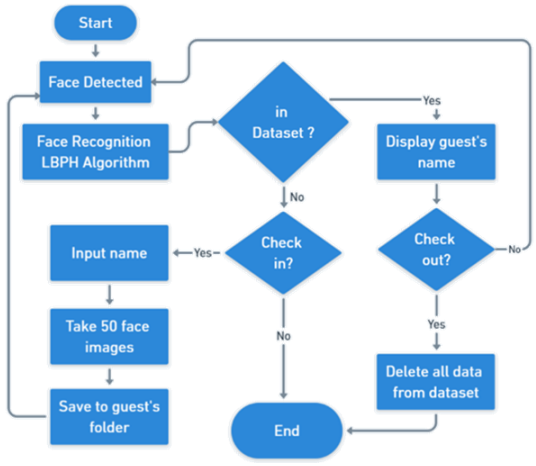


Figure 1. Flowchart System

### *2.1 Facial Recognition*

The detailed flow of the facial recognition embedded in the robot can be seen in Figure 2. The program design system starts by converting the previously colored (RGB) image into a grayscale using Equation 1.

$$Grayscale = \frac{R+G+B}{3} \quad (1)$$

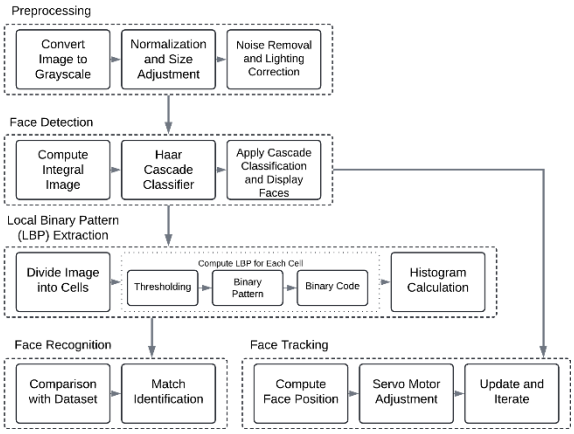


Figure 2. Facial processing workflow

In a colored image, each pixel consists of three-color components: red (R), green (G), and blue (B). To obtain the grayscale value, this equation calculates the average of these three-color components. In other words: $R$ is the red intensity value, $G$ is the green intensity value, and $B$ is the blue intensity value. The equation sums the red, green, and blue intensity values, then divides the total by 3 to obtain the grayscale value [17]. This value represents the brightness of the pixel, where 0 is black and 255 is white on an 8-bit scale. This process is essential because a grayscale image reduces data complexity by using only one-color channel, making it more efficient for processing.

After conversion, resizing or normalization and adjustment of the size are performed to ensure the image is on a uniform scale, avoiding differences in face size that could affect detection accuracy. To resize an image mathematically, interpolation formulas can be used. There are several interpolation methods available, but one of the most common is bilinear interpolation. Suppose we want to resize an image from $W \times H$ to $W' \times H'$. A point on the original image at coordinates (x, y) will be mapped to a new point on the resized image. The new coordinates on the resized image are ($x'$, $y'$) as shown in Equations 2 and 3.

$$x' = \frac{x}{W} \times W' \quad (2)$$

$$y' = \frac{y}{H} \times H' \quad (3)$$

The point ($x'$, $y'$) on the new image may not align exactly with the pixel grid of the resized image. To obtain the pixel value at this point, we use bilinear interpolation. Suppose $(x_1, y_1)$, $(x_2, y_1)$, $(x_1, y_2)$ and $(x_2, y_2)$ are the coordinates of the pixels on the original image surrounding the point ($x'$, $y'$), with pixel intensities $I(x_1, y_1)$, $I(x_2, y_1)$, $I(x_1, y_2)$, and $I(x_2, y_2)$. Horizontal interpolation is performed using Equations 4 and 5.

$$I_{x',y1} = I(x_1, y_1) \times (1 - \alpha) + I(x_2, y_1) \times \alpha \quad (4)$$

$$I_{x',y2} = I(x_1, y_2) \times (1 - \alpha) + I(x_2, y_2) \times \alpha \quad (5)$$

α is $x'$ - $x1$ (horizontal projection). Vertical interpolation is performed using Equation 6.

$$I(x', y'), \ = I_{x',y1} \times (1 - \beta) + I_{x',y2} \times \beta \quad (6)$$

β is $y'$ - $y1$ (vertical projection).

Then, the noise removal and lighting correction stages are performed to improve image quality, ensuring that the image is free from disturbances that could affect facial recognition accuracy. A Gaussian filter is used to smooth the image and reduce noise by averaging the intensity of surrounding pixels. This filter uses a Gaussian function, given by Equation 7.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (7)$$

$\sigma$ is the standard deviation of the Gaussian distribution. This function assigns greater weight to pixels close to the center of the filter window and smaller weight to pixels further away. Convolving the image with the Gaussian kernel results in a smoother image by reducing noise and unwanted fine details.

Contrast adjustment aims to enhance the difference between light and dark areas in the image to make details more distinct [12]. This is achieved by modifying the pixel intensity scale using Equation 8.

$$I'(x, y) = \alpha I(x, y) + \beta \quad (8)$$

$\alpha$ is the contrast factor and $\beta$ is the brightness offset. A contrast factor $\alpha$ greater than 1 will increase the

contrast, making dark areas darker and light areas lighter, while the offset $\beta$ shifts the pixel intensity range, increasing or decreasing overall brightness. By applying this adjustment, the image will have optimal contrast, making features more prominent and easier to analyze.

The next step is to compute the integral image, which helps speed up the computation process for feature recognition [18]. The integral image, also known as the summed-area table, is a highly useful technique for accelerating feature calculations in image processing. This technique allows for very efficient computation of the sum of pixels in a rectangular area. The integral image Iint(x, y) at position (x, y) is defined as the sum of all pixel intensities in the top-left part of the image up to that position [19], calculated using Equations 9.

$$I_{int}(x,y) = \sum_{i=0}^{x}\sum_{j=0}^{y} I(i,j) \quad (9)$$

$I(i,j)$ is the pixel intensity at the position $(i,j)$ in the original image. Once the integral image is computed, the sum of pixels in a rectangular area with the top-left corner at $(x_1, y_1)$ and the bottom-right corner at $(x_2, y_2)$ can be calculated using Equation 10.

$$sum(x_1,y_1,x_2,y_2) = I_{int}(x_2,y_2) - I_{int}(x_1-1,y_2) - I_{int}(x_2,y_2-1) + I_{int}(x_1-1,y_1-1) \quad (10)$$

This equation allows for the calculation of the sum of pixels in constant time $O(1)$, which is highly efficient compared to traditional methods. With the integral image, you can accelerate the computation process for feature recognition, such as in face detection, making the image analysis process much faster and more efficient.

Using the Haar Cascade Classifier faces in an image are identified through filtering techniques that search for specific patterns or features on the face [20]. Haar features work by comparing bright and dark rectangular areas in the image, where the pixel intensity in the brighter area is subtracted from the pixel intensity in the darker area. This feature is expressed in Equation 11.

$$F = \sum_{area\ putih} I(x,y) - \sum_{area\ hitam} I(x,y) \quad (11)$$

$I\ (x,\ y)$ is the pixel intensity at position $(x,\ y)$. To accelerate the computation of Haar features, the integral image is used, which enables efficient calculation of pixel intensity sums in rectangular areas. The method is then trained using the AdaBoost algorithm, which selects the most important Haar features from the numerous available features. Each feature is used to form a weak classifier, and the combination of these weak classifiers creates a strong classifier that is more accurate in detecting objects. The Haar Cascade also employs a cascading structure, where the image is processed through multiple stages of classifiers. If an area fails to be detected as an object at an early stage, it is immediately discarded, thus speeding up the detection process. This method is known for its efficiency in image processing because it combines Haar features with the AdaBoost algorithm and cascade structure, making it one of the fast and accurate facial detection methods.

Once a face is detected, the system displays the face using Cascade Classification and also calculates the position of the face captured by the camera. At this stage, the system will adjust the servo motors on the robot to direct the camera, which is positioned where the robot's eyes are, to always face the detected face. The servo motors are used to control the movement along the x and y axes, ensuring that the user's face remains at the center of the robot's camera. This process is repeated to continuously monitor the face.

The Local Binary Pattern Histogram (LBPH) method is an approach used for facial recognition by utilizing local texture patterns. The process begins by converting the face image to grayscale, which simplifies the data being processed by focusing on the light intensity of each pixel [10]. Then, the Local Binary Pattern (LBP) algorithm is applied, which compares each central pixel with its surrounding neighbors in a 3x3 block as shown in Equation 12. A binary pattern is generated based on the pixel intensity comparison, where a value of 1 is assigned if the neighboring pixel is greater than or equal to the central pixel, and a value of 0 if it is smaller.

$$LBP = \sum_{i=0}^{k-1} s(N_i - P)\,.\,2^i \quad (12)$$

$s(x)\ is\ 1$ if $x \geq 0$, and $s(x)\ is\ 0$ if $x < 0$, $N_i$ is the intensity of the i-th neighboring pixel, $P$ is the intensity of the central pixel.

After the LBP pattern is computed for all pixels in the image, the face image is divided into several small blocks, and the histogram of LBP values is computed for each block. This histogram represents the frequency of occurrence of LBP patterns within that block, and the histograms from each block are then combined to form a global histogram that describes the entire face [21] as shown in Equation 13.

$$H(j) = \sum_{x,y} I(LBP(x,y) = j) \quad (13)$$

$H(j)$ is the histogram value for the LBP pattern -$j$, $I$ is the indicator function that equals 1 if $LBP(x,y) = j$, and 0 otherwise, $x$ and $y$ are the pixel coordinates within the block.

After the new face histogram is calculated, the next step is to compare it with the histogram stored in the database. This comparison is often done using Euclidean distance, which measures the distance between two face histograms [22] as shown in Equation 14.

$$d(H_1,H_2) = \sqrt{\sum_{i=1}^{n}(H_1(i) - H_2(i))^2} \quad (14)$$

$H_1$ and $H_2$ are the histograms of two face images, $n$ is the number of elements in the histogram.

If the Euclidean distance between the new image histogram and one of the histograms in the database is small, the new face is recognized as a registered face. This process allows for LBPH-based facial recognition

to be performed accurately and efficiently, despite its simplicity.

### *2.2 Robot Development*

This project involves the development of an interactive robot consisting of several key components such as a Raspberry Pi 3, camera, keyboard, monitor, Arduino Uno, and servo motors (see Figure 3). The Raspberry Pi 3 acts as the processing hub, running the facial recognition program using a camera placed in the robot's eye section. This camera is used to detect and track the user's face in real-time. The keyboard functions as the user data input, while the monitor provides a visual interface that displays the results of facial recognition or other interactions.

The Arduino Uno is used as the main controller to operate two servo motors that control the X and Y axes of the robot's head. These servos enable the robot's head to follow the user's movements, ensuring that the camera remains directed at the user's face. In the robot's physical design, as shown in the diagram, the robot's head is made from 3D-printed materials with the servo system embedded inside (see Figure 4). These servos enable mechanical movement of the eyes and head to follow the user's movements, creating a more natural interaction.

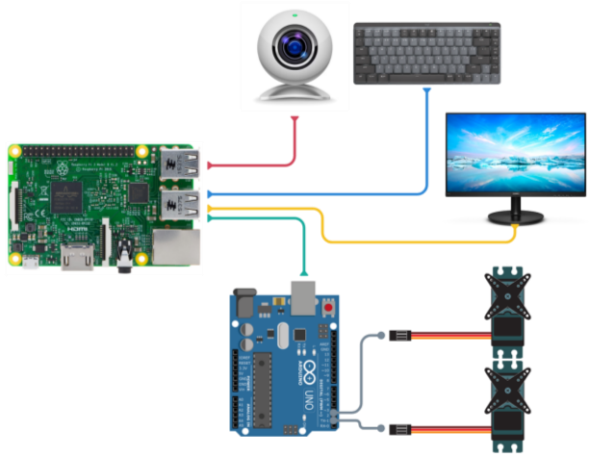

Figure 3. Schematic Robot


Figure 4. Robot Design

The camera mounted in one of the robot's eyes is connected to the facial recognition system on the Raspberry Pi. After detecting a face, the system sends a signal to the Arduino to control the servo motors, enabling the robot's head and eyes to follow the user's movements. With this design, the robot is capable of mimicking human movements in a more interactive and responsive manner, providing a more realistic experience in facial recognition and tracking.

## 3. Results and Discussions

In this section, we will explore the results of various tests and analyses conducted using methods appropriate to their respective scales.

### *3.1 Face Recognition Result*

The first test involves converting an initially colored (RGB) image into a black-and-white (grayscale) image to simplify image processing and improve computational efficiency.

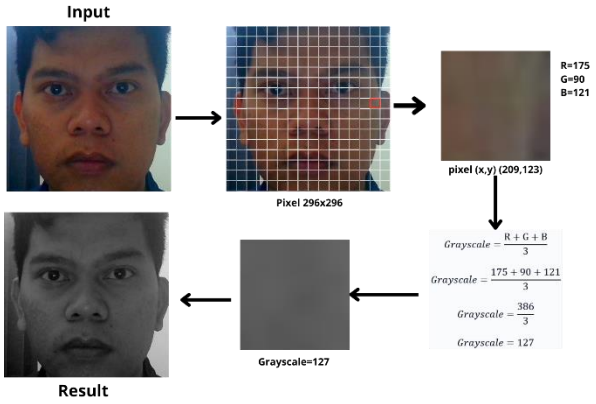


Figure 5. RGB to Grayscale

The conversion process from RGB to grayscale for a 296x296 pixel image begins by extracting the RGB values from each pixel in the image (Figure 5). For example, at a pixel with coordinates (x, y) = (209, 123), the RGB values are (175, 90, 121), where 175 is the red channel value, 90 is the green channel value, and 121 is the blue channel value. The first step in the conversion is to calculate the grayscale value for this pixel using the standard conversion formula, which takes into account the relative contribution of each color channel. Equation (1) will be used for this calculation.

After the calculation, the grayscale value obtained for the pixel (x, y) = (209, 123) is approximately 127. As a result, in the new grayscale image, the pixel at the same coordinates (209, 123) will have an intensity value of 127. This process is repeated for each pixel in the 255x255 image, resulting in a grayscale image of the same dimensions, where each pixel has a single grayscale intensity value instead of the three RGB color values. The process continues for each pixel until every pixel in the color image is converted to grayscale. In this way, the initially 255x255 RGB image is transformed into a grayscale image of the same dimensions, but with color information replaced by grayscale intensity values (see Figure 6).

 

The next step is to resize the image, where the previously grayscale-converted image of 296x296 pixels is resized to 255x255 pixels. To resize the image from 296x296 pixels to 255x255 pixels manually, we can use interpolation methods, one of which is a nearest neighbor interpolation. For each point $(x, y)$ in the original image, we calculate the new position $(x', y')$ using Equations 2 and 3.

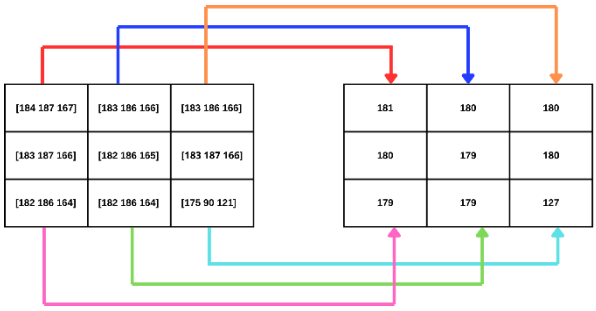


Figure 6. the RGB conversion to a 3x3 Grayscale value

After calculating $x'$ and $y'$, we use bilinear interpolation to determine the pixel intensity value at $(x', y')$. If (129, 129) is the nearest coordinate in the new pixel grid, we use the four nearest pixels from the original image to compute the intensity value at this position. Since the coordinates $(x', y') = (129.39, 129.39)$ do not fall exactly on the new pixel grid, we need to use the intensity values from the four nearest pixels in the original image: $(x1, y1) = (129,129)$.

$(x2, y1) = (130,129)$
(x1,y2)=(129,130)
$(x2, y2) = (130,130)$

Let the pixel intensities at each of these points be:

$I(x1, y1) = I(129,129)$
$I(x2, y1) = I(130,129)$
$I(x1, y2) = I(129,130)$
$I(x2, y2) = I(130,130)$

We start with horizontal interpolation to obtain the intensity value between $(x1, y1)$ and $(x2, y1)$ at the coordinate $(x', y1)$, and between $(x1, y2)$ and $(x2, y2)$ at the coordinate $(x', y2)$

$$I(x', y1) = I(x1, y1) \times (1 - \alpha) + I(x2, y1) \times \alpha$$
$$I(x', y2) = I(x1, y2) \times (1 - \alpha) + I(x2, y2) \times \alpha$$

Where α=x′−x1=129.39−129=0.39

Thus, we can calculate:

$$I(x', y1) = I(129,129) \times (1 - 0.39) + I(130,129) \times 0.39$$

$$I(x', y2) = I(129,130) \times (1 - 0.39) + I(130,130) \times 0.39$$

After obtaining $I(x', y1)$ and $I(x', y2)$, we perform vertical interpolation to calculate the final intensity value at the coordinate $(x', y')$:

$$I(x', y') = I(x', y1) \times (1 - \beta) + I(x', y2) \times \beta$$

Where:

$\beta = y' - y1 = 129.39 - 129 = 0.39$
$I(x', y') = I(x', y1) \times (1 - 0.39) + I(x', y2) \times 0.39$

If $I(x', y1) = a; I(x', y2) = b$

Then:

$$I(x', y') = a \times (1 - 0.39) + b \times 0.39$$

$$I(x', y') = a \times 0.61 + b \times 0.39$$

Thus, we can determine the pixel intensity value at $(x', y')$, which is the result of interpolation from the four nearest pixels in the original image. This process is applied to every pixel in the new image, resulting in a resized image from 296x296 to 255x255 pixels with smoother transitions.

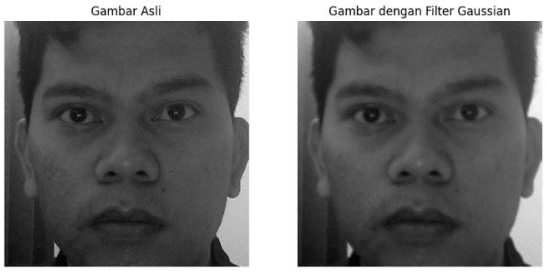


Figure 7. Filter Gaussian

From the results of the image that has been converted to grayscale and resized to 255x255 pixels, the next step is noise removal from the image using the Gaussian filter equation shown in Figure 7. Here, we demonstrate the calculation using a 3x3 pixel example we previously illustrated,

$$\text{Image} = \begin{bmatrix} 181 & 180 & 180 \\ 180 & 179 & 180 \\ 179 & 179 & 127 \end{bmatrix}$$

$$Kernel = \begin{bmatrix} 0.0613 & 0.1221 & 0.0613 \\ 0.1221 & 0.2442 & 0.1221 \\ 0.0613 & 0.1221 & 0.0613 \end{bmatrix}$$

To calculate the pixel value resulting from the Gaussian filter at the center position (1,1) of the image, we perform a convolution with the Gaussian kernel:

$$Result = \sum_{i=-1}^{1} \sum_{j=-1}^{1} (\text{Image}[i + 1, j + 1] \times \text{Kernel}[i + 1, j + 1])$$

With the updated pixel values, the pixel value resulting from the Gaussian filter for the center position (1,1) on the given 3x3 grayscale image is approximately 171.60. In this way, you can calculate the resulting pixel values for the entire image if the image is larger, by sliding the Gaussian kernel to each pixel position and performing the convolution calculation.

Next is contrast adjustment, which aims to enhance the difference between light and dark areas in the image to make the details clearer.

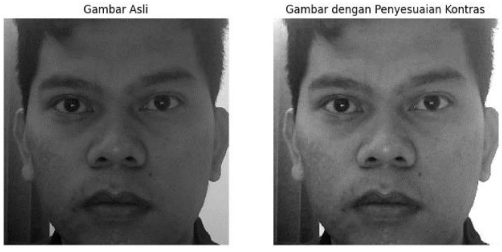


Figure 8. contrast adjustment

To achieve the results as shown in Figure 8, the author applied contrast adjustment using equation (8), where we want to set the contrast with α=1.5 (contrast factor) and β=0 (brightness offset). This results in updated values for each pixel.

After the contrast adjustment with α=1.5 and β=0, the resulting image is:

$$Image = \begin{bmatrix} 181 & 180 & 180 \\ 180 & 179 & 180 \\ 179 & 179 & 127 \end{bmatrix}$$

At this step, each pixel value has been adjusted according to the given contrast factor and clipped to the 0-255 range. This is the result of the contrast adjustment, which enhances the difference between light and dark areas in the image shown in Table 1.

Table 1. The result of the contrast adjustment on 3x3 pixel image

| Pixel (x, y) | Original Intensity | α × I(x, y) | Offset β | Result I'(x, y) | Clamping Result |
|---|---|---|---|---|---|
| (0, 0) | 181 | 271.5 | 0 | 271.5 | 255 |
| (0, 1) | 180 | 270 | 0 | 270 | 255 |
| (0, 2) | 180 | 270 | 0 | 270 | 255 |
| (1, 0) | 180 | 270 | 0 | 270 | 255 |
| (1, 1) | 179 | 268.5 | 0 | 268.5 | 255 |
| (1, 2) | 180 | 270 | 0 | 270 | 255 |
| (2, 0) | 179 | 268.5 | 0 | 268.5 | 255 |
| (2, 1) | 179 | 268.5 | 0 | 268.5 | 255 |
| (2, 2) | 127 | 190.5 | 0 | 190.5 | 190 |

In the integral image process, the result of contrast adjustment will be calculated using Equation 9 which produces Figure 9 as shown in Table 2.

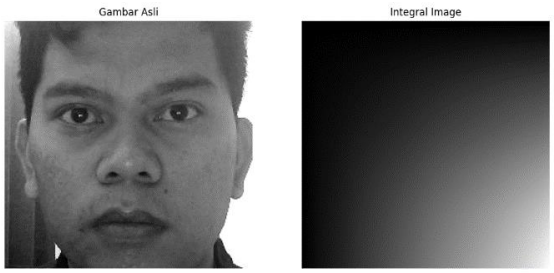


Figure 9. Integral Image

$$Integral\ Image = \begin{bmatrix} 255 & 255 & 255 \\ 255 & 255 & 255 \\ 255 & 255 & 190 \end{bmatrix}$$

Table 2. Integral image calculation results on 3x3 pixels

| Pixel (x,y) | Intensity I'(x,y) | Integral (x,y) | Calculation |
|---|---|---|---|
| (0, 0) | 255 | 255 | 255 |
| (0, 1) | 255 | 510 | 255+255 |
| (0, 2) | 255 | 765 | 510+255 |
| (1, 0) | 255 | 510 | 255+255 |
| (1, 1) | 255 | 1020 | 510+510 |
| (1, 2) | 255 | 1275 | 765+255 |
| (2, 0) | 255 | 765 | 510+255 |
| (2, 1) | 255 | 1275 | 765+510 |
| (2, 2) | 190 | 1720 | 1275+190 |

Integral Image is a cumulative table that stores the sum of pixels from the top-left corner to a specific point in the image, allowing for more efficient calculation of Haar-Like features. Haar-Like features are used to detect certain patterns in an image, such as edges, corners, or lines, by calculating the difference between dark and light areas in rectangular shapes. By using the Integral Image, Haar-Like features are calculated quickly without requiring access to individual pixels. Once the Haar-Like features are computed, a Haar Cascade Classifier is applied to detect faces by testing specific patterns that have been learned. If the recognized pattern matches a face pattern, a bounding box is drawn around the face in the image (Figure 10). This process enables accurate and effective face detection in grayscale images by utilizing specific features and efficient data structures.

Extracted features for face at (x=14, y=14):

[1.359e+03 0.000e+00 2.000e+00 0.000e+00 0.000e+00 2.000e+00 1.000e+00
4.000e+00 0.000e+00 4.000e+00 4.000e+00 2.000e+00 3.000e+00 7.000e+00
1.400e+01 1.100e+01 1.100e+01 2.100e+01 2.000e+01 2.900e+01 3.800e+01
4.700e+01 6.000e+01 5.500e+01 6.800e+01 8.500e+01 9.200e+01 1.140e+02…

Figure 10. result of the haar cascade classifier

Once the face has been detected in the previous process, the next step is to divide the face into small blocks, such as blocks sized 3x3 pixels. Each of these small blocks will be processed to calculate what we call the Local Binary Pattern (LBP), as shown in Figure 11. The program works by taking one pixel in the center of the block as the central pixel, and then comparing the intensity value of this central pixel with its neighbors around the 3x3 block.

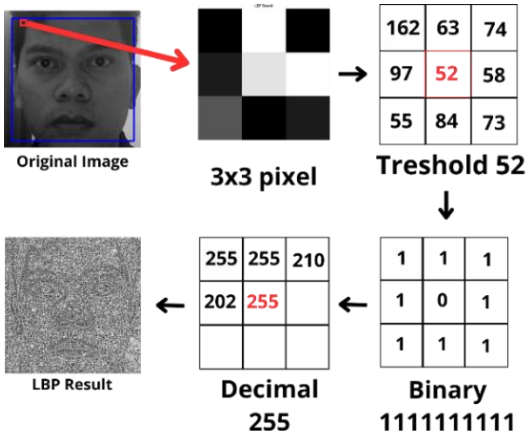


Figure 11. LBP operation

If the neighbor's value is greater than or equal to the central pixel, we assign a value of 1. Conversely, if the neighbor is less than the central pixel, we assign a value of 0. These values then form a binary pattern. For example, if we have a central pixel with an intensity of 52, and its neighbors have higher values like 162, 170, 180, and so on, we will get a binary pattern of all 1s.

This binary pattern is then converted into a decimal value. In this case, the binary 11111111 will be converted to 255 in decimal.

After the decimal values are obtained from each central pixel throughout the face, these values are organized into a histogram representing the overall facial features. This histogram contains the frequency of binary patterns found in the face blocks. This histogram becomes the feature vector that can be used to match someone's face with stored facial data.

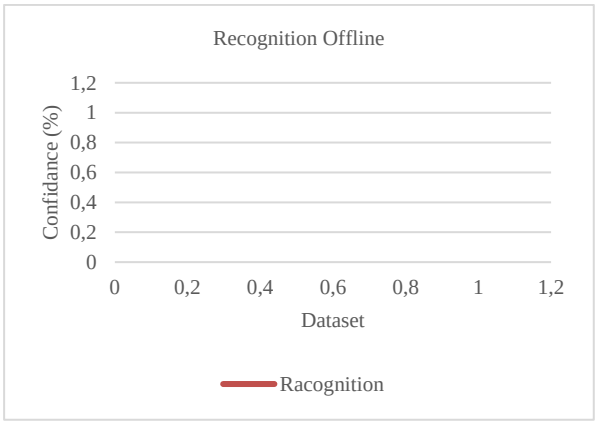


Figure 12. The test results for face recognition offline.

The graph in Figure 12 shows the results of face recognition testing using the LBPH (Local Binary Patterns Histograms) algorithm in two different scenarios: offline and real-time recognition. In the offline testing, one face data is selected from a dataset containing 50 faces and then matched one by one with all the faces in the dataset shown in Table 3. The graph shows that the confidence level in offline face recognition varies between 80.5% and 85.5% with an average of 83.6%. The fluctuations observed reflect variations in the quality or similarity of faces in the dataset, with some significant drops indicating that certain faces may have weaker or different features compared to others in the dataset.

Table 3. The test results for face recognition offline.

| No | Dataset Image | Confidence (%) | Time Taken (S) |
|---|---|---|---|
| 1 | face_0,jpg | 83,43 | 0,014 |
| 2 | face_1,jpg | 84,33 | 0,0152 |
| 3 | face_2,jpg | 84,13 | 0,014 |
| 4 | face_3,jpg | 84,41 | 0,015 |
| 5 | face_4,jpg | 84,06 | 0,015 |
| 6 | face_5,jpg | 84,87 | 0,0141 |
| 7 | face_6,jpg | 84,21 | 0,0163 |
| 8 | face_7,jpg | 83,3 | 0,0171 |
| 9 | face_8,jpg | 84,23 | 0,0153 |
| 10 | face_9,jpg | 83,8 | 0,0161 |
| 11 | face_10,jpg | 85,08 | 0,0183 |
| 12 | face_11,jpg | 84,59 | 0,0162 |
| 13 | face_12,jpg | 84,59 | 0,0161 |
| 14 | face_13,jpg | 85,09 | 0,0152 |
| 15 | face_14,jpg | 84,88 | 0,0151 |
| 16 | face_15,jpg | 84,58 | 0,0161 |
| 17 | face_16,jpg | 83,96 | 0,014 |
| 18 | face_17,jpg | 84,95 | 0,0132 |
| 19 | face_18,jpg | 85,27 | 0,0171 |
| 20 | face_19,jpg | 84,71 | 0,0141 |
| 21 | face_20,jpg | 84,47 | 0,0142 |
| 22 | face_21,jpg | 84,52 | 0,0141 |
| 23 | face_22,jpg | 84,38 | 0,0141 |
| 24 | face_23,jpg | 84,11 | 0,0141 |
| 25 | face_24,jpg | 81,55 | 0,0132 |
| 26 | face_25,jpg | 81,11 | 0,0142 |
| 27 | face_26,jpg | 82,34 | 0,0151 |
| 28 | face_27,jpg | 83,42 | 0,0142 |
| 29 | face_28,jpg | 82,91 | 0,0143 |
| 30 | face_29,jpg | 83,34 | 0,0134 |
| 31 | face_30,jpg | 82,37 | 0,0141 |
| 32 | face_31,jpg | 82,31 | 0,0143 |
| 33 | face_32,jpg | 83,5 | 0,0141 |
| 34 | face_33,jpg | 82,36 | 0,0151 |
| 35 | face_34,jpg | 83,41 | 0,014 |
| 36 | face_35,jpg | 83,17 | 0,0142 |
| 37 | face_36,jpg | 82,64 | 0,0152 |
| 38 | face_37,jpg | 82,37 | 0,0133 |
| 39 | face_38,jpg | 82,62 | 0,0142 |
| 40 | face_39,jpg | 82,66 | 0,0141 |
| 41 | face_40,jpg | 82,59 | 0,0142 |
| 42 | face_41,jpg | 83,33 | 0,0131 |
| 43 | face_42,jpg | 82,04 | 0,0141 |
| 44 | face_43,jpg | 83,44 | 0,0142 |
| 45 | face_44,jpg | 83,92 | 0,0142 |
| 46 | face_45,jpg | 83,86 | 0,0152 |
| 47 | face_46,jpg | 83,1 | 0,017 |
| 48 | face_47,jpg | 83,55 | 0,0141 |
| 49 | face_48,jpg | 83 | 0,0143 |
| 50 | face_49,jpg | 83,28 | 0,0142 |
| Average | | 83,6028 | 0,014712 |

Meanwhile, in real-time testing, the face captured every second is directly matched with the 50 face data in the same dataset shown in Table 4. The graph in Figure 13 on the right shows confidence levels ranging from 72% to 81% with an average of 77.67%. The greater fluctuations in this graph indicate that changes in conditions during image capture, such as lighting, position, or facial expression, greatly affect the accuracy of recognition. Overall, offline recognition appears more stable compared to real-time, although both suggest that the quality of facial images and environmental conditions are crucial to achieving more accurate and consistent recognition results.

Table 4. The test results for face recognition in real-time.

| No | Dataset Image | Confidence (%) | Time Taken(s) |
|---|---|---|---|
| 1 | face_0,jpg | 75,51 | 0,0205 |
| 2 | face_1,jpg | 78,75 | 0,02 |
| 3 | face_2,jpg | 78,26 | 0,019 |
| 4 | face_3,jpg | 79,62 | 0,018 |
| 5 | face_4,jpg | 77,96 | 0,018 |
| 6 | face_5,jpg | 78,87 | 0,0188 |
| 7 | face_6,jpg | 78,69 | 0,021 |

| No | Dataset Image | Confidence (%) | Time Taken(s) |
|---|---|---|---|
| 8 | face_7,jpg | 77,81 | 0,0205 |
| 9 | face_8,jpg | 77,97 | 0,022 |
| 10 | face_9,jpg | 78,87 | 0,02 |
| 11 | face_10,jpg | 77,23 | 0,021 |
| 12 | face_11,jpg | 77,64 | 0,019 |
| 13 | face_12,jpg | 78,02 | 0,019 |
| 14 | face_13,jpg | 76,56 | 0,019 |
| 15 | face_14,jpg | 75,9 | 0,0195 |
| 16 | face_15,jpg | 77,07 | 0,021 |
| 17 | face_16,jpg | 77,78 | 0,0165 |
| 18 | face_17,jpg | 77,06 | 0,02 |
| 19 | face_18,jpg | 78,06 | 0,019 |
| 20 | face_19,jpg | 77,4 | 0,0195 |
| 21 | face_20,jpg | 76,5 | 0,022 |
| 22 | face_21,jpg | 76,42 | 0,021 |
| 23 | face_22,jpg | 76,61 | 0,022 |
| 24 | face_23,jpg | 76,52 | 0,0209 |
| 25 | face_24,jpg | 73,26 | 0,0207 |
| 26 | face_25,jpg | 73,54 | 0,0195 |
| 27 | face_26,jpg | 75,12 | 0,019 |
| 28 | face_27,jpg | 77,15 | 0,017 |
| 29 | face_28,jpg | 76,61 | 0,02 |
| 30 | face_29,jpg | 78,38 | 0,029 |
| 31 | face_30,jpg | 77,54 | 0,045 |
| 32 | face_31,jpg | 78,89 | 0,0185 |
| 33 | face_32,jpg | 78,58 | 0,019 |
| 34 | face_33,jpg | 77,49 | 0,0204 |
| 35 | face_34,jpg | 78,48 | 0,019 |
| 36 | face_35,jpg | 77,03 | 0,019 |
| 37 | face_36,jpg | 79,27 | 0,021 |
| 38 | face_37,jpg | 77,55 | 0,019 |
| 39 | face_38,jpg | 78,75 | 0,021 |
| 40 | face_39,jpg | 78,97 | 0,02 |
| 41 | face_40,jpg | 78,88 | 0,0175 |
| 42 | face_41,jpg | 78,12 | 0,0815 |
| 43 | face_42,jpg | 78,76 | 0,019 |
| 44 | face_43,jpg | 79,22 | 0,052 |
| 45 | face_44,jpg | 77,91 | 0,018 |
| 46 | face_45,jpg | 78,45 | 0,019 |
| 47 | face_46,jpg | 77,2 | 0,02 |
| 48 | face_47,jpg | 78,36 | 0,023 |
| 49 | face_48,jpg | 79,17 | 0,019 |
| 50 | face_49,jpg | 79,91 | 0,019 |
| Average | | 77,6734 | 0,022246 |

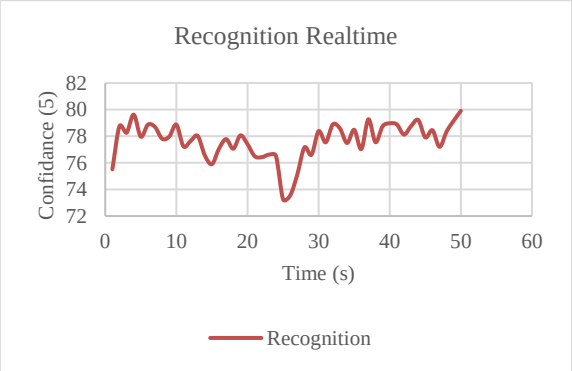


Figure 13. The test results for face recognition realtime.

### *3.2 Face Tracker Evaluation*

Table 5 shows the results of an experiment with a robot designed to follow a face. The face position is displayed in the form of pixel coordinates X and Y taken from the camera. For example, in the first row, the face position is detected at X = 302 and Y = 212. The camera captures the face at that point. The robot's servo then adjusts its position based on the face movement. In the first example, the servo does not need to move because the face is already centered. However, in the second row, the face shifts to X = 459 and Y = 204, so the servo changes its position to X = 137° and Y = 79° to follow the movement to the upper right.

Table 5. The test results for face tracker.

| Position | Face Position (pixel) | | Servo Position (°) | | Movement | | Time (s) |
|---|---|---|---|---|---|---|---|
| | x | y | x | y | x | y | |
|  | 302 | 212 | 110 | 90 | C | C | 0 |
|  | 459 | 204 | 137 | 79 | R | U | 2.314 |
|  | 399 | 278 | 132 | 96 | L | D | 1.945 |
|  | 518 | 342 | 139 | 105 | R | D | 2.952 |
|  | 219 | 302 | 91 | 94 | L | U | 3.561 |
| | 391 | 159 | 119 | 82 | R | U | 3.013 |

The robot also moves to align with the face's direction. In the second example, the face is detected on the right and upward, so the robot moves to the right and upward to adjust its position. The time taken to adjust the face back to the center position is also recorded. In the second row, the robot takes 2.314 seconds to make this adjustment. Additionally, images captured by the camera are included for each position change, providing visual confirmation that the robot is accurately following the face movement. This table illustrates how the robot reacts to face movements, adjusts its servo positions, and records the time needed to center the face back on the screen.

### *3.3 Discussions*

The results of testing the face recognition system using the Local Binary Patterns Histogram (LBPH) algorithm indicate that this method can provide an adequate level of accuracy in both offline and real-time modes,

although several factors influence system performance. In offline testing, an average accuracy of 83.6% shows that LBPH can recognize faces well when the database facial data has high quality and uniformity. Accuracy fluctuations ranging from 80.5% to 85.5% suggest variability in face recognition, which could be caused by differences in perspective or lighting in the dataset. Real-time testing, with an average accuracy of 77.67%, reveals that environmental dynamics such as changes in lighting and facial expressions directly affect the results of face recognition shown in Table 6.

Table 6. Results of the comparison between offline and real-time face recognition.

| Mode | Minimum Accuracy (%) | Maximum Accuracy (%) | Average Accuracy (%) |
|---|---|---|---|
| Offline | 80.5 | 85.5 | 83.6 |
| Real-time | 72.0 | 81.0 | 77.67 |

This indicates that the system is more sensitive to changing real-time conditions, which is a common challenge in direct face recognition applications. The difference in processing times between offline and online modes is primarily due to the computational load involved in real-time image acquisition and processing. In offline mode, pre-captured images are already stored in memory, allowing the system to focus solely on recognition tasks, leading to more consistent processing times. However, in online mode, the system must simultaneously handle image acquisition, pre-processing (grayscale conversion, noise reduction, and contrast adjustment), and recognition. Factors such as varying lighting conditions and rapid facial movements can add additional processing time, resulting in increased variability in response times.

On the other hand, the servo movements in the robot designed to follow the user's face showed positive results, with response times ranging from 1.945 seconds to 3.561 seconds. This demonstrates that the robot can adjust its position quickly and accurately, enabling more natural interaction between the robot and the user. Although these results are satisfactory, there is room for further improvements, particularly in optimizing the system to handle more challenging environmental conditions and to speed up the robot's response time. Overall, this study successfully demonstrates the great potential of implementing face recognition systems in receptionist robots, though there is still room for refinement to address various practical challenges in the field.

## 4. Conclusions

This research successfully developed a memory-efficient face recognition system using the Local Binary Patterns Histogram (LBPH) algorithm implemented on a Raspberry Pi-based receptionist robot. The test results show that the LBPH algorithm is capable of providing good face recognition accuracy, with an average accuracy of 83.6% in offline mode and 77.67% in real-time mode. Nonetheless, the system is still affected by environmental changes such as lighting and facial expressions, which affect recognition accuracy. Additionally, the designed robot is capable of accurately following the user's facial movements, with a fairly fast servo response time of between 1.945 seconds and 3.561 seconds, supporting responsive and natural interaction with the user. Although the results are satisfactory, this research identifies several areas for improvement, such as further optimization for handling dynamic environmental conditions and improving the robot's response speed. Overall, the developed system shows great potential for implementation in interactive and efficient receptionist robot applications.

## Acknowledgements

This research has received funding from the Directorate of Research, Technology and Community Service, Directorate General of Higher Education, Research and Technology, Ministry of Education, Culture, Research and Technology (grant agreement 104/E5/PG.02.00.PL/2024, 1100/LL2/KP/PL/2024). The authors thank Universitas Bina Darma for its support and facilities.

## References

[1] N. Li *et al.*, "Chinese Face Dataset for Face Recognition in an Uncontrolled Classroom ENVIRONMENT," *IEEE Access*, vol. 11, 2023, doi: 10.1109/ACCESS.2023.3302919.

[2] R. He, J. Cao, L. Song, Z. Sun, and T. Tan, "Adversarial Cross-Spectral Face Completion for NIR-VIS Face Recognition," *IEEE Trans Pattern Anal Mach Intell*, vol. 42, no. 5, 2020, doi: 10.1109/TPAMI.2019.2961900.

[3] J. Tomášik *et al.*, "The Potential of AI-Powered Face Enhancement Technologies in Face-Driven Orthodontic Treatment Planning," *Applied Sciences 2024, Vol. 14, Page 7837*, vol. 14, no. 17, p. 7837, Sep. 2024, doi: 10.3390/APP14177837.

[4] P. C. P. Neto, J. R. Pinto, F. Boutros, N. Damer, A. F. Sequeira, and J. S. Cardoso, "Beyond Masks: On the Generalization of Masked Face Recognition Models to Occluded Face Recognition," *IEEE Access*, vol. 10, 2022, doi: 10.1109/ACCESS.2022.3199014.

[5] H. Yu, Y. Wang, Y. Tian, H. Zhang, W. Zheng, and F. Y. Wang, "Social Vision for Intelligent Vehicles: From Computer Vision to Foundation Vision," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 11, pp. 4474–4476, Nov. 2023, doi: 10.1109/TIV.2023.3330870.

[6] W. Sun, X. Min, D. Tu, S. Ma, and G. Zhai, "Blind Quality Assessment for in-the-Wild Images via Hierarchical Feature Fusion and Iterative Mixed Database Training," *IEEE Journal on Selected Topics in Signal Processing*, vol. 17, no. 6, pp. 1178–1192, Nov. 2023, doi: 10.1109/JSTSP.2023.3270621.

[7] K. Panetta *et al.*, "A Comprehensive Database for Benchmarking Imaging Systems," *IEEE Trans Pattern Anal Mach Intell*, vol. 42, no. 3, pp. 509–520, Mar. 2020, doi: 10.1109/TPAMI.2018.2884458.

[8] Y. Zhong *et al.*, "Dynamic Training Data Dropout for Robust Deep Face Recognition," *IEEE Trans Multimedia*, vol. 24, pp. 1186–1197, 2022, doi: 10.1109/TMM.2021.3123478.

[9] C. WANG, "A Comparative Statement for the Facial Recognition Algorithm Via Eigenfaces and the Local Binary Patterns Histograms Algorithm," *i-manager's Journal on Software Engineering*, vol. 15, no. 4, p. 1, Apr. 2021, doi: 10.26634/JSE.15.4.18429.

[10] I. Al Saidi, M. Rziza, and J. Debayle, "A novel texture descriptor: Homogeneous Rotated Local Binary Pattern (HRLBP)," *2020 10th International Symposium on Signal,*

 

*Image, Video and Communications, ISIVC 2020*, Apr. 2021, doi: 10.1109/ISIVC49222.2021.9487538.

[11] S. Chakraborty, S. K. Singh, and K. Kumar, "Facial Biometric System for Recognition Using Extended LGHP Algorithm on Raspberry Pi," *IEEE Sens J*, vol. 20, no. 14, pp. 8117–8127, Jul. 2020, doi: 10.1109/JSEN.2020.2979907.

[12] Z. Al-Ameen, H. N. Saeed, and D. K. Saeed, "Fast and Efficient Algorithm for Contrast Enhancement of Color Images," *Review of Computer Engineering Studies*, vol. 7, no. 3, p. 60, Sep. 2020, doi: 10.18280/RCES.070303.

[13] D. Shehada, A. Turky, W. Khan, B. Khan, and A. Hussain, "A Lightweight Facial Emotion Recognition System Using Partial Transfer Learning for Visually Impaired People," *IEEE Access*, vol. 11, pp. 36961–36969, 2023, doi: 10.1109/ACCESS.2023.3264268.

[14] H. Gong, L. Chen, C. Li, J. Zeng, X. Tao, and Y. Wang, "Online Tracking and Relocation Based on a New Rotation-Invariant Haar-Like Statistical Descriptor in Endoscopic Examination," *IEEE Access*, vol. 8, pp. 101867–101883, 2020, doi: 10.1109/ACCESS.2020.2994440.

[15] Y. Guo, Q. Xu, Y. Su, S. J.-I. Access, and undefined 2020, "Visibility detection based on the recognition of the preceding vehicle's taillight signals," *ieeexplore.ieee.orgY Guo, Q Xu, Y Su, S JiangIEEE Access, 2020•ieeexplore.ieee.org*, Accessed: Sep. 14, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9257398/

[16] S. Liu, J. Chen, … S. R. C. and S. for V., and undefined 2019, "A new multi-focus image fusion algorithm and its efficient implementation," *ieeexplore.ieee.orgS Liu, J Chen, S RahardjaIEEE Transactions on Circuits and Systems for Video Technology, 2019•ieeexplore.ieee.org*, Accessed: Sep. 14, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8653405/

[17] K. Chen, T. Yi, Q. L.-I. S. P. Letters, and undefined 2021, "Lightqnet: Lightweight deep face quality assessment for risk-controlled face recognition," *ieeexplore.ieee.orgK Chen, T Yi, Q LvIEEE Signal Processing Letters, 2021•ieeexplore.ieee.org*, Accessed: Sep. 14, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9528058/

[18] D. Kim, J. Hyun, and B. Moon, "Memory-efficient architecture for contrast enhancement and integral image computation," *2020 International Conference on Electronics, Information, and Communication, ICEIC 2020*, Jan. 2020, doi: 10.1109/ICEIC49074.2020.9051296.

[19] A. H. Diab, I. T. Ahmed, and W. M. Jasim, "Analyze the Existing Contrast Enhancement Algorithms and Image Quality Assessment: Survey," *AIP Conf Proc*, vol. 3009, no. 1, Feb. 2024, doi: 10.1063/5.0193832/3265139.

[20] G. Ghosh and K. S. Swarnalatha, "A Detail Analysis and Implementation of Haar Cascade Classifier," pp. 341–359, 2022, doi: 10.1007/978-981-16-3342-3_28.

[21] W. El-Tarhouni, A. Abdo, and A. Elmegreisi, "Feature fusion using the Local Binary Pattern Histogram Fourier and the Pyramid Histogram of Feature fusion using the Local Binary Pattern Oriented Gradient in iris recognition," in *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering, MI-STA 2021 - Proceedings*, 2021. doi: 10.1109/MI-STA52233.2021.9464473.

[22] D. Loktev, A. Loktev, R. Stepanov, M. Faisal, and E. M. Zamzami, "Comparative Analysis of Inter-Centroid K-Means Performance using Euclidean Distance, Canberra Distance and Manhattan Distance," *J Phys Conf Ser*, vol. 1566, no. 1, p. 012112, Jun. 2020, doi: 10.1088/1742-6596/1566/1/012112.

 