IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 15, No. 4, August 2026, pp. 3903~3918
ISSN: 2252-8938, DOI: 10.11591/ijai.v15.i4.pp3903-3918

A novel embedded approach to face recognition using multi-threaded controller based on weightless neural network

Ahmad Zarkasi[1], Hadipurnawan Satria[2], Anggina Primanita[2], Deris Stiawan[1]
[1]Department of Computer Engineering, Faculty of Computer Science, Universitas Sriwijaya, Palembang, Indonesia
[2]Department of Informatics Engineering, Faculty of Computer Science, Universitas Sriwijaya, Palembang, Indonesia

Article Info

Article history:

Received Nov 1, 2025
Revised Jul 13, 2026
Accepted Jul 21, 2026

Keywords:

Center scan recognition
Embedded system
Face recognition
Multi-thread controller
Real-time processing
Weightless neural network

ABSTRACT

This research presents a high efficiency embedded face recognition system based on the weightless neural network-face recognition algorithm (WNN-FRA) integrated with a multi-thread controller to enhance execution time and recognition accuracy under limited hardware resources. The system implements a center-scan feature alignment model to address resolution discrepancies between reference and input facial images. The multi-threaded architecture divides processing into three concurrent threads front, left, and right facial orientations each handling approximately 20 facial patterns. Experimental evaluation on a dataset of 60 facial images demonstrated a maximum recognition accuracy of 96.83% and an average execution time ranging from 16 to 34 milliseconds per dataset, confirming real-time performance. Comparative analysis shows that the multi-threaded approach reduced the execution time by over 67% compared to single-thread processing 0.09 second vs. 0.271 second, while maintaining balanced workload distribution across threads. Memory analysis revealed that the entire system required only 24 KB from the available 512 KB flash capacity, indicating efficient resource utilization. The results confirm that integrating WNN-FRA with multi-threading provides a robust, low-cost, and scalable solution for real-time facial pattern recognition in embedded environments.

Corresponding Author:

Ahmad Zarkasi
Department of Computer Engineering, Faculty of Computer Science, Universitas Sriwijaya
Palembang, Indonesia
Email: zarkasi98@gmail.com

1. INTRODUCTION

The primary challenge in image pattern processing on embedded systems lies in the inherent limitations of their computational resources. This challenge becomes increasingly complex when the system is required to combine multiple analytical methods for processing the same object [1]–[3]. Such constraints often hinder the development and implementation of facial pattern recognition systems on devices employing single-chip controller architectures [4]–[6]. To ensure the effectiveness of face-pattern processing on embedded platforms, two key criteria must be met: fast execution time and flexibility in the classification process [7]. Several previous studies have demonstrated that the weightless neural network (WNN) approach, particularly the Wilkie, Stonham, and Aleksander recognition device (WiSARD) model [8]–[10], provides an efficient solution to these challenges. The WiSARD method utilizes a storage and classification mechanism in which facial pattern data are distributed across multiple random-access memory (RAM) nodes organized within a set of RAM discriminators, each responsible for identifying a specific pattern class. The WiSARD architecture consists of several discriminators operating in parallel to evaluate input images and determine their correspondence to the represented class. Its generative property makes WiSARD particularly relevant

for applications in open set recognition, where the system must identify previously unseen data that are not part of the training set [11]–[13]. Research on open set recognition has shown that WiSARD exhibits a superior ability to represent the distinct characteristics of data distributions compared to conventional classification approaches. The principal advantage of the WiSARD method lies in its ability to produce smoother matching contours, more evenly distributed measurement data, and more representative similarity values between input and reference patterns [14], [15]. These characteristics make WiSARD highly suitable for implementation on embedded systems, as all computational processes are executed directly within the RAM units, significantly reducing processing load and execution time [16], [17].

Embedded-based face pattern processing is a technical approach that integrates multiple processor components designed to support each other, both in terms of software and hardware [18]–[20]. The main objective of this integration is to enable the system to operate in parallel and simultaneously, as well as to distribute the computational workload between hardware and software components to achieve more efficient facial recognition processing. Several previous studies [21]–[23] have shown that facial patterns can be grouped into multiple discriminators, each serving as a processing unit for the facial recognition task. To accelerate the identification process, each facial group is assigned a specific label or binary code for instance, the front face (10), right face (01), and left face (11). When the input data corresponds to a front-face label (10), the algorithm performs recognition only within the front-face library, while the right and left-face libraries are bypassed [24]. Although this strategy effectively reduces execution time, the algorithm presents certain limitations, as not all facial data are processed simultaneously. If all face patterns were processed concurrently, the execution time would increase up to threefold. Moreover, the recognition accuracy for right and left faces is not necessarily lower than that of the front face. This limitation often arises from challenges in accurately determining the region of interest (ROI), particularly for faces oriented to the right or left. Consequently, the system may sometimes recognize right or left faces accurately even when the input corresponds to a front-face pattern, and vice versa, resulting in a certain degree of uncertainty in recognition outcomes. To address these challenges, a data-processing mechanism capable of operating in parallel and simultaneously within a single-chip controller is required. The most appropriate approach for this purpose is the implementation of a multithreading mechanism [25], [26], which enables the facial pattern recognition process to be executed concurrently across multiple independent execution threads, thereby minimizing processing time without compromising system accuracy.

Literally, a thread is defined as the smallest unit of a processor that can be executed within a process. A thread encompasses several processor resources, including a program counter, a stack pointer, and a dedicated data area for its own stack, allowing the processor to independently execute branching instructions [27]–[29]. Operationally, a thread can function in three primary modes: i) sequential execution, ii) interruption allowing the processor to switch to another thread, and iii) concurrent execution with other threads within the same process, a mechanism commonly known as multithreading. Multithreading is a processing technique in which a single process within an application is divided into multiple threads that can run in parallel [30]–[32]. This approach enhances computational efficiency since each thread can handle different parts of a task simultaneously. Consequently, the implementation of multi-threading serves as a potential solution to overcome execution time constraints in embedded-based systems. In this research, a WNN method combined with the center scan algorithm is proposed as the primary approach for face-pattern recognition. The entire recognition process is implemented within a multi-thread controller to ensure parallel and efficient execution. The main objective of this research is to develop a novel algorithm that performs facial data matching based on the central position of an image by first determining its optimal center point. This approach is expected to produce a simple, fast, and accurate computational process for face-pattern recognition, enabling the core data to be processed completely and correctly. Furthermore, the processed data are grouped into memory cells for subsequent analysis, generating high-quality information that accurately represents the degree of similarity between the input pattern and the reference pattern. Overall, this study aims to develop a face pattern detection system characterized by shorter execution time, higher recognition accuracy, and a hardware prototype design that is simple, cost-effective, and efficient for embedded implementation.

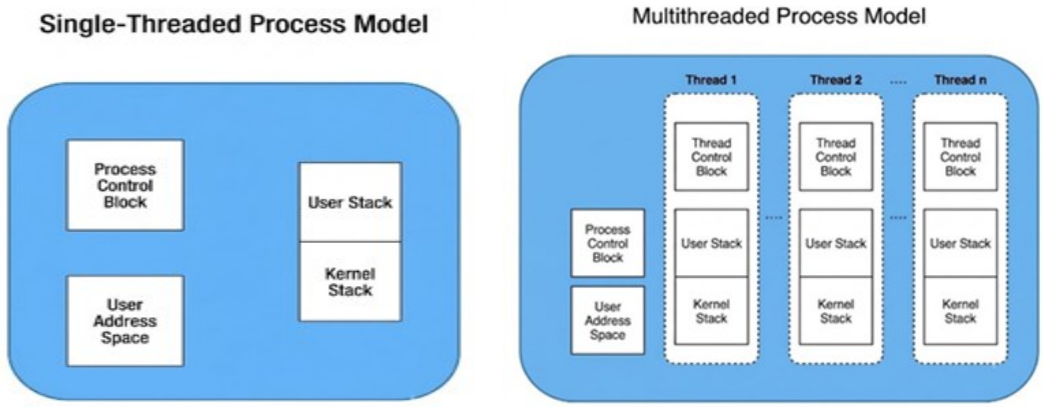
2. MATERIAL AND CONTROLLER

2.1. Multi-threaded controller

As a component of computation, a thread carries the essential processing resources required for program execution. These include the program counter, which stores the address of the next instruction to be executed; the stack pointer, which points to the memory location within the stack; and a dedicated data area for the stack, which enables the execution of branching instructions and subroutine calls. With these attributes, a thread is capable of performing program execution in a structured and flexible manner. Based on

its operational principles, a thread can function in three primary conditions: first, it can be executed sequentially following the instruction flow; second, it can be temporarily paused or interrupted to allow the processor to switch execution to another thread; and third, it can run concurrently with other threads within the same process, a concept known as multithreading [33], [34]. Multithreading itself is a programming and process management technique in which a single process executing an application is divided into multiple threads that can operate in parallel or simultaneously. This approach enhances processing efficiency by distributing workloads across several execution units. Consequently, the implementation of multithreading is often regarded as one of the most effective solutions for improving application performance, accelerating system responsiveness, and optimizing processor resource utilization.

Multi-threading fundamentally refers to a process management capability that enables the provision and control of multiple concurrent execution paths within a single process. In other words, a single process can contain several threads running in parallel, allowing the system to enhance processor utilization efficiency. Unlike traditional approaches where each process contains only one execution thread and the concept of threading is not yet implemented, this conventional model is referred to as single threaded. Both processes are illustrated in Figure 1, which represents the threaded process model. The left side of Figure 1(a) illustrates the single threaded model, in which process management supports only one user process with a single thread, as exemplified by Microsoft-disk operating system (MS-DOS). Conversely, the multi-threaded approach, illustrated on the right side of Figure 1(b), has been widely adopted by modern process management systems that integrate multithreading capabilities to improve performance, efficiency, and overall system responsiveness [35]–[37]. In the multi-threaded model, a process is regarded as the primary unit of resources responsible for managing and coordinating the execution of multiple threads within it.



(a) (b)

Figure 1. Threaded process model for (a) single threaded and (b) multi-threaded

In this research, threads are utilized to execute the face-pattern recognition process in parallel. In previous research, the facial pattern recognition process employed an encoding mechanism, where facial patterns consisting of multiple orientations such as front, right, left, top, and bottom were assigned specific recognition codes. The purpose of this mechanism is to enable the face recognition algorithm (FRA) to determine which facial orientation serves as the input. Through this approach, the detected code directs the system to identify the input data according to its corresponding facial-pattern group. The advantage of this mechanism lies in its ability to reduce the execution time of the recognition process, as only a specific subset of facial patterns is processed rather than the entire dataset from index 1 to n. However, a major challenge arises when the system fails to accurately determine the precise facial region, leading to overlapping recognition between adjacent facial orientations. For instance, a front face orientation with an angle between 30° and 45° may still be recognized as a front face, as illustrated in Figure 2.

Figure 2 illustrates two facial image orientations, namely the front and left face positions. The front orientation is presented in Figures 2(a) and 2(c) with an angle of 0°, while the left face orientation is shown in Figures 2(b) and 2(d) with an approximate angle of 40°. Figures 2(a) and 2(b) represent the original facial images, whereas Figures 2(c) and 2(d) display the facial images extracted from the region designated as the ROI. Ideally, in case Figure 2(d), the face should already be categorized within the left face region. Nevertheless, in practice, the front-face classifier is still capable of detecting and recognizing the pattern accurately within the frontal category. This indicates the presence of overlapping areas between facial

orientations. The most effective approach to ensure comprehensive recognition of all facial orientations is to perform a complete scan of the facial area. However, this would cause the system to lose real-time performance and significantly increase recognition time, thereby affecting the overall efficiency of the hardware in performing face-pattern recognition. Figure 3 presents the proposed architecture of the multi-threaded controller developed to address this issue.

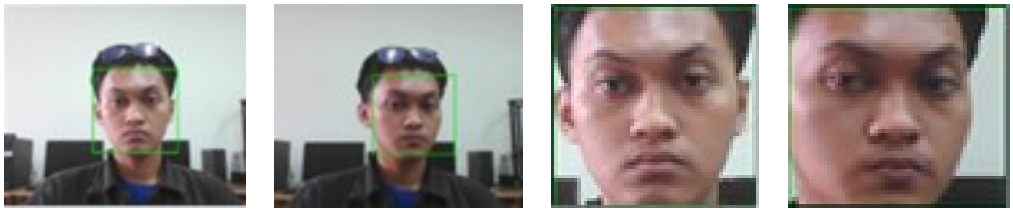


Figure 2. Overlapping facial image recognition between front and left view: (a) original image (0°), (b) original image (40°), (c) ROI (0°), and (d) ROI (40°)

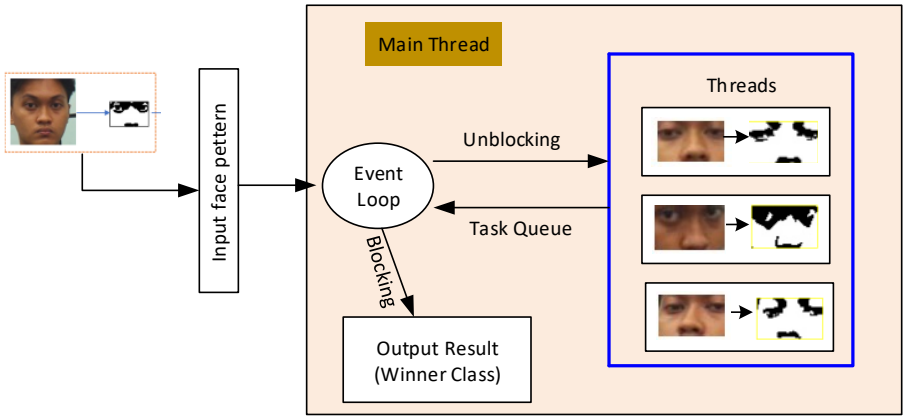


Figure 3. Proposed multi-threaded controller architecture

This research delineates the functional architecture of the multi-thread controller into three primary components: data extraction, data analysis, and data integration, as illustrated in Figure 3. In the data extraction stage, the main thread is responsible for managing dual data streams originating from the input face patterns, which are subsequently placed into a task queue. These data streams consist of extracted facial features that serve as identifiers, where facial area information is represented in a binary format to enable further processing. By implementing a multi-threaded controller architecture, the data execution process becomes more efficient, as computational workloads can be distributed and executed in parallel. This approach contrasts with the single-thread model, which requires longer execution time due to its inherent limitations in handling multiple data streams simultaneously.

In the data analysis stage, each thread functions as a RAM discriminator that actively retrieves data from the task queue for analysis. The task queue serves to organize and display the sequence of results obtained from the pattern recognition process within the RAM discriminators and to determine the winner class pattern. Once the similarity accuracy of the recognized pattern is obtained, the result is transmitted back to the result queue as the winner class. Each thread represents a pattern function that operates independently when activated, thereby enhancing the controller's performance in reading and processing data streams.

Within the multi-threaded architecture, the use of the WNN method, which utilizes RAM as the primary data-processing medium, simplifies the computational process of face-pattern recognition. The data integration stage focuses on the role of the main thread, which is responsible for managing, processing, and integrating data originating from the various threads within the system. The integration process is carried out through a blocking function mechanism, which allows the system to wait until all required data are available before merging them. This approach enhances the quality of data processing results by ensuring that integration occurs in a controlled and accurate manner, thereby producing consistent and complete outputs.

Furthermore, it provides greater flexibility in determining the format, structure, and presentation strategy of the data, whether for visualization purposes or for transmission to other systems.

2.2. Weightless neural network face recognition algorithm design

The learning process in this study was conducted by developing a WNN-FRA that implements a center-scan pattern recognition model. The WNN-FRA method is designed to process feature-extraction data obtained from the face detection stage. These data are represented as facial patterns captured from three image orientations: front, left, and right. In the WNN-FRA method, the learning process is performed using RAM, where the facial pattern data are represented in a binary format. This binary representation is chosen to align with the fundamental principles of the WNN, which relies on a simple memory structure yet possesses the capability to perform efficient pattern classification. Consequently, each facial pattern obtained from feature extraction can be systematically mapped into the WNN memory for both training and testing purposes.

In the training phase, each RAM discriminator must first be initialized by setting all memory locations in the RAM to a logical state of 0. This initialization process ensures that no default values remain in memory that could affect the learning outcome. Once the initialization stage is complete, the system selects a training pattern represented by a binary vector of $(X \cdot n)$ bits. This binary representation serves as the input address used to activate specific memory locations within the RAM. For each training pattern provided, a logical value of 1 is stored in the RAM memory location corresponding to the address determined by the input pattern. This procedure is repeated iteratively for all available training patterns, allowing the RAM discriminator to gradually construct a memory map that represents the relationship between input patterns and their corresponding memory addresses.

After the entire training process is completed, the contents of the RAM memory consist of a combination of logical values 0 and 1. A logical value of 1 indicates that the corresponding memory location has been addressed by one of the training patterns, whereas a logical value of 0 signifies that the location remains unused. At this stage, the RAM memory can be regarded as having stored the initial data patterns required for the subsequent pattern recognition phase. Next, the outputs from each RAM node are aggregated by a summation device (Σ). This component functions to sum all outputs from the RAM nodes within a single RAM discriminator, and the resulting total is then divided by the overall number of RAM nodes (K). Consequently, the obtained output value reflects the proportion of RAM nodes that produce a logical 1 for a given input vector X. This data serves as a measure of the similarity between the tested input pattern and the reference patterns stored in memory during the training process. Through this mechanism, the RAM discriminator is capable of distinguishing input patterns based on the distribution of logical 1 value formed during training.

The complete architecture of the WNN-FRA method developed in this research is illustrated in Figure 4. The architecture demonstrates the flow of facial pattern data from the feature extraction stage through the training phase, data storage in memory, and finally to the face recognition process. The sequential stages of the WNN-FRA learning process are as follows: i) determine the number of input vectors obtained from facial feature extraction, both those stored in the database and those used as primary inputs; ii) divide the database input vectors into three main groups right, front, and left; iii) process these data within several discriminator classes under the multithreading mechanism; iv) perform simultaneous recognition of the data across each discriminator class using the thread-based mechanism; and v) determine the optimal value from the discriminator classes to identify the winner class. This architecture provides a clear operational framework illustrating how facial data are processed and classified efficiently through parallelized learning and recognition mechanisms based on the WNN-FRA.

Figure 4 illustrates that the learning process consists of two main components: reference patterns and input patterns. The reference patterns represent facial images that have undergone feature extraction, beginning with the acquisition of red, green, blue (RGB) facial data and followed by the face detection stage. Once the face is detected, a feature detection process is carried out to identify key facial components such as the eyes, nose, and mouth. This process produces a new image that captures only a specific facial pixel region, for instance, of size n × m pixels. The resulting feature image is then converted into a grayscale image and subsequently into a binary image. The binary image is normalized into data values of 0 and 1. After the facial feature patterns are formed, these binary patterns serve as input vectors for the WNN method and are stored across 20 RAM nodes distributed within three RAM discriminators. To enable simultaneous processing of input-pattern recognition, the three RAM discriminators are configured to function as a multi-thread controller (Thread 1, Thread 2, and Thread 3) under the main thread block.

In the input pattern stage, the process is similar to that of the reference pattern the key difference lies in the fact that the reference patterns are stored within the dataset, whereas the input pattern serves as the pattern to be recognized during the learning process. The input RGB pattern, representing facial features, undergoes a series of extraction stages until it is converted into binary data. The resulting binary data take the

form of a matrix corresponding to the pixel dimensions of the input facial pattern. Once the input pattern is prepared, the data are forwarded to the recognition process within the comparator block, where the pattern is compared simultaneously across all threads. Each thread performs 20 recognition iterations, allowing the system to obtain both the pattern accuracy level and the execution time performance for each facial pattern.

Since the input pattern image data are captured within a dynamically moving system, the number of pixels obtained varies across samples. This variation affects the pattern recognition accuracy, as the pixel dimensions of the reference pattern differ from those of the input pattern. To address this issue, the present study employs a center-scan recognition model. Figure 5 illustrates the design of the center-scan model. The center-scan recognition algorithm is designed to improve the accuracy level achieved by the previous algorithm. This approach emphasizes achieving balance between the reference pattern and the input pattern, particularly when discrepancies arise due to differences in memory capacity or image resolution between the reference pattern (dataset) and the input pattern. Such discrepancies occur because the input pattern is acquired from dynamic facial positions, which may result in smaller or larger image dimensions. The center-scan method minimizes the difference in memory capacity observed in previous algorithms by locating the central point during the initial checking process.

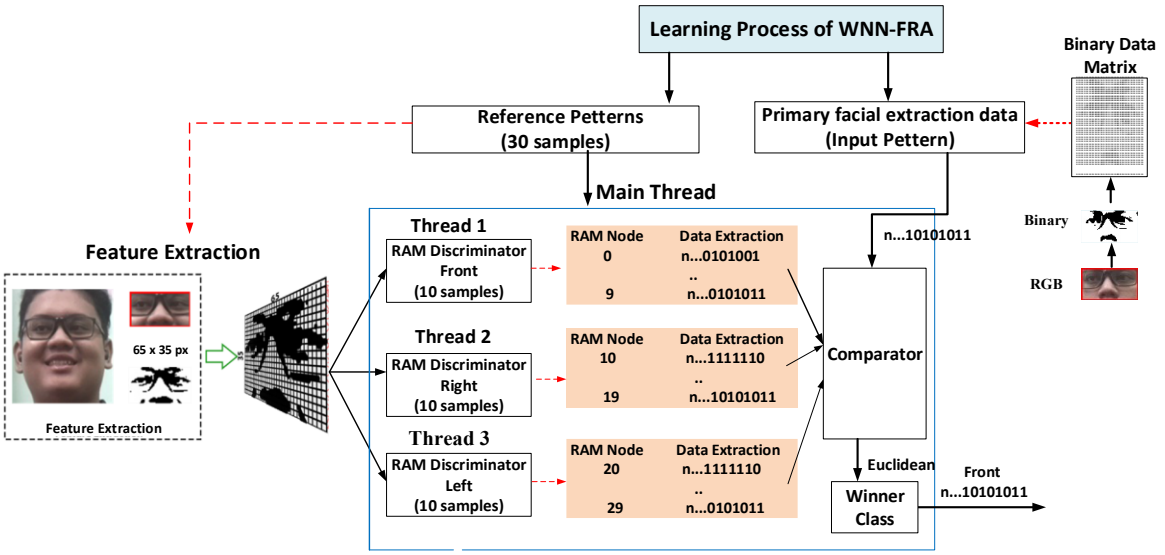


Figure 4. WNN-FRA architecture design

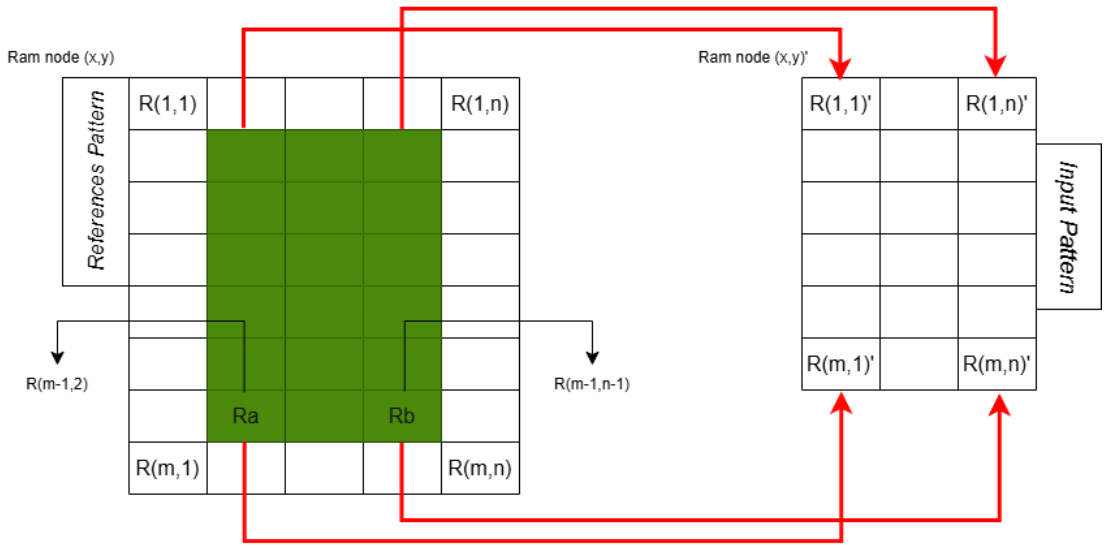


Figure 5. Center scan model design

As illustrated in Figure 5, the input pattern has a smaller data capacity than the reference pattern, specifically with two fewer columns in the input matrix. Before the checking process begins, the algorithm first determines the initial and final positions for verification. Accordingly, the initial checking position is defined at RAM node points R (1, n-6) to R (m, n-6), which are equivalent to R (1,1)' to R (m,1)'. The final checking position is set at RAM node points R (1, n-1) to R (m, n-1), which correspond to R (1, n)' to R (m, n)'. This rule also applies inversely when the opposite condition occurs. The sequence of operations in the center-scan recognition algorithm is as follows; i) if the memory capacity (resolution) of the input pattern data is equal to that of the reference pattern, the comparator directly performs the verification process based on the predefined initial and final positions, ii) if the memory capacity of the input pattern is smaller than that of the reference pattern, the algorithm first identifies the initial and final points of the reference pattern before performing the checking process. The comparison is then conducted according to the defined memory capacity; and iii) conversely, if the input pattern has a larger memory capacity than the reference pattern, the algorithm first determines the initial and final points of the input pattern, after which the data comparison is executed based on the corresponding memory capacity.

3. RESULT AND DISCUSSION

3.1. Feature extraction result

The feature extraction process consists of several stages, beginning with face detection. This stage starts by capturing facial images as input data [1]. The images acquired from the camera are not processed directly but first undergo a preprocessing stage. In this stage, the color image RGB is converted into a grayscale image. The purpose of this conversion is to reduce complexity, simplify and accelerate computation, while still preserving the essential information required for face detection. Subsequently, the system identifies regions that potentially contain facial objects. A scanning process is performed across the entire image to locate sub-windows that are likely to include a face. Each sub-window is then analyzed based on previously extracted features. This approach allows computational resources to be focused only on relevant areas, thereby improving both processing efficiency and detection accuracy. The final stage employs the cascade classifier algorithm, which operates through multiple testing stages that progressively eliminate image regions dissimilar to facial patterns. In the initial stage, the testing parameters are kept simple to ensure that potential face candidates are not missed. In subsequent stages, the criteria become increasingly selective, retaining only regions that strongly match facial characteristics. This hierarchical filtering mechanism effectively reduces false detections and yields more accurate predictions of facial positions within the image [2], [3].

After the face is successfully detected in the camera image, the system determines the ROI, the specific portion of the image that contains the user's facial area. The ROI is derived from the coordinates obtained during the previous Haarcascade detection process, consisting of the x and y positions as well as the width and height values of the detected facial area in each frame. The size of the ROI may vary depending on the proximity of the face to the camera. To maintain data consistency during the recognition stage, the ROI is resized to 88 × 88 pixels, representing 88 pixels horizontally (from the left to the right side of the face) and 88 pixels vertically (from the forehead to the chin). The next step involves converting the ROI image from RGB format to grayscale. Following this, the system applies a thresholding process using an adaptive threshold value that adjusts according to the average light intensity within the ROI, ensuring stable conversion results despite lighting variations. The thresholding process produces a binary image, which is then normalized using a resize method to a fixed size of 60 × 60 pixels. This normalization aims to standardize the facial image dimensions used as input, thereby simplifying subsequent stages of data storage, processing, and recognition. Table 1 presents the experimental results of the face detection stage.

Table 1. Face detection result

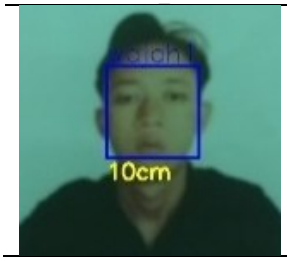
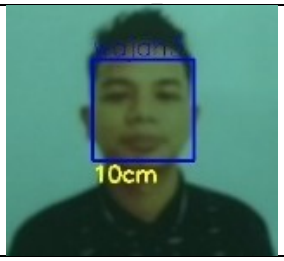
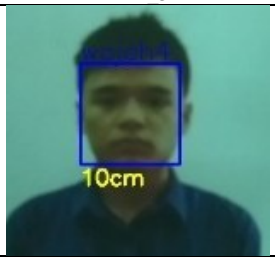
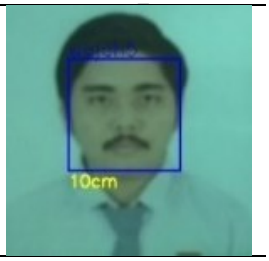
Face detection			
Face 1	Face 2	Face 3	Face 4
			

Table 1 presents the results of testing conducted using four different facial samples. There are four facial patterns: Face_1, Face_2, Face_3, and Face_4. Face_1 represents the user's facial pattern data, which is later used as the dataset in the learning process, whereas Face_2, Face_3, and Face_4 are facial patterns employed for testing during the recognition phase. Image acquisition was carried out at several distances—10 cm, 20 cm, and 30 cm although the table only displays the results obtained at a distance of 10 cm. The purpose of this face detection test is to ensure that the system functions according to its design and to evaluate the accuracy level in recognizing facial objects under varying conditions. The testing process was performed using both still images and video recorded directly through the camera. Furthermore, testing was conducted with variations in facial orientation, including frontal, left-side, and right-side views. These variations were chosen to assess the system's performance when encountering changes in environmental conditions and object positioning. Once the facial pattern was successfully detected, the system displayed a blue bounding box around the identified face within the frame. This visualization, as shown in Table 1, serves as an indication of the successful execution of the face detection process.

The next stage is the facial feature extraction process, which serves as a crucial step in a face recognition system, as it aims to obtain information representing the unique characteristics of each individual. In this study, the Haarcascade classifier algorithm is employed to detect facial features within the image. In addition to identifying the presence of a face, the algorithm also produces coordinate information in the form of a bounding box, a rectangular boundary indicating the exact location of the face in each camera frame. As illustrated in Figure 6, which depicts the feature extraction process, once the facial position is detected through the bounding box in the raw image, the system determines the ROI, the specific portion of the image containing the relevant facial features for further analysis, with a pixel dimension of 88 × 88. Determining the ROI is essential, as it allows computational resources to be focused on the areas containing high informational value while reducing unnecessary processing on irrelevant image regions. In this research, the ROI serves as the primary input for transforming visual data into a digital representation that can be processed by the recognition system. In addition to functioning as the input for feature extraction, the ROI size is also utilized to estimate the relative distance between the face and the camera. Generally, a larger ROI indicates that the face is closer to the camera, and conversely, a smaller ROI corresponds to a greater distance. Once the data are obtained, the next step is to convert them into binary format.

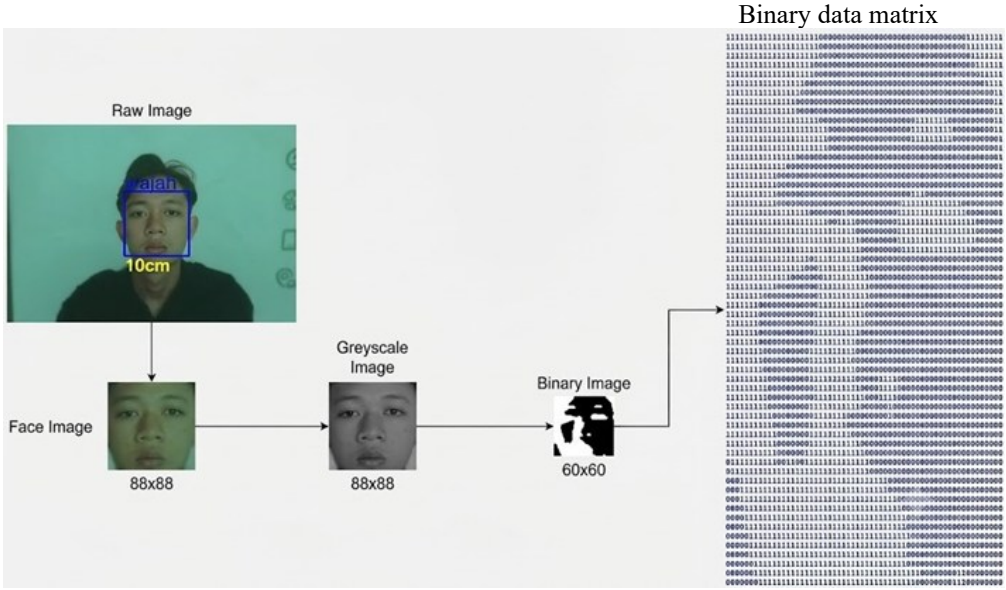


Figure 6. Feature extraction result

The conversion of images into binary form is a crucial stage, as it simplifies data processing within the microcontroller system. This is achieved by transforming the visual representation into a fundamental digital format binary. The conversion process begins by transforming the ROI image from RGB format into grayscale. Subsequently, the system applies a thresholding method with an adaptive threshold value that adjusts according to the average light intensity within the ROI, ensuring that the conversion process remains stable despite variations in illumination conditions. The result of the thresholding process is a binary image, which is then normalized in size through a resize operation to a fixed resolution of 60 × 60 pixels.

This normalization ensures consistent facial data dimensions used as input, thereby facilitating the processes of storage, transmission, and recognition. The final outcome of the feature extraction stage in this study is a facial pattern represented as a binary image. This binary data also serves as the dataset for the subsequent pattern recognition process.

3.2. Memory requirement analysis

Before designing the facial pattern dataset, a memory requirement analysis was conducted on the single-chip microcontroller system [7], [8]. This analysis aimed to determine the memory capacity needed to efficiently store and manage facial pattern data [10]. This step is essential because the available resources in a microcontroller are inherently limited; therefore, precise calculations are required to ensure that the system operates optimally without exceeding the capacity of the hardware used. The microcontroller utilized in this study is the ARM® 32-bit Cortex®-M4, which features a floating-point unit (FPU), a maximum CPU frequency of 84 MHz, 512 KB of Flash memory, and 96 KB of SRAM. The limited memory capacity presents one of the primary challenges in implementing an embedded-based face recognition system, as the storage and processing of digital images demand relatively large memory space. Consequently, an optimization strategy is required, one that involves modifying and managing data allocation in memory to ensure efficient resource utilization. In this research, the facial pattern dataset is implemented within the main thread, which operates with three active threads [25], [26]. Each thread is responsible for processing 20 facial patterns stored in memory. Specifically, Thread 1 handles front-face patterns, Thread 2 processes left-face patterns, and Thread 3 is dedicated to right-face patterns. With this structure, the recognition process can be executed in parallel, thereby improving execution efficiency while maintaining system stability. The results of the system memory requirement calculations are presented in Table 2, which serves as the basis for evaluating memory efficiency and resource optimization in this embedded face recognition system.

Table 2. Memory requirement

Threads	Number of face patterns (a = 20)	Binary data matrix (pixel)	Number of pixels (P = a × pixel)	Number of memories (n = p/8) byte	Total memory requirement (KB)
Thread 1	Front	60 × 60	72,000	9,000	9
Thread 2	Left	60 × 50	60,000	7,500	7.5
Thread 3	Right	60 × 50	60,000	7,500	7.5

After determining the total number of facial patterns to be used in the dataset, a comprehensive memory requirement calculation was carried out to ensure that the allocated capacity does not exceed the maximum limit available on the microcontroller. This calculation process involved several key parameters, including the size of each facial pattern, the number of active threads, and the distribution of data between flash memory and SRAM. Based on the analysis results, the total memory required to store 60 facial patterns in the dataset is 24 KB, consisting of 9 KB for front-face patterns, 7.5 KB for right-face patterns, and 7.5 KB for left-face patterns. These findings indicate that the ROM memory capacity of the microcontroller still has 488 KB available. The next stage involves constructing the facial pattern dataset based on the acquired image data. Image acquisition is performed using a Raspberry Pi camera (RaspiCam) mounted on the robot's neck section. The placement of the camera in this position enables direct image capture of the user's face with an aligned viewing angle—covering the frontal, left-side, and right-side perspectives. Each captured facial image represents a distinct individual facial pattern and serves as a data source for building the facial pattern dataset. This image acquisition process is designed to be flexible, allowing the generation of visual data directly in digital format. Consequently, the dataset construction process can be performed efficiently without requiring additional conversion steps.

The data captured by the RaspiCam, after being converted into binary images, are subsequently used as the representation of the extracted facial patterns. The image representation in binary format is illustrated in Figure 6, which shows the structure of the binary data matrix as the final output of the image conversion and visual data storage processes within the system. The resulting binary data are then saved in a text file with a .txt extension to allow further processing in the subsequent stages. The text file containing the binary matrix is processed using a C-based program, which converts it into a dataset.cpp file functions as a library within the system. In the dataset.cpp file, the binary data are declared and stored using the program memory (PROGMEM) directive, ensuring that the data are placed in the flash memory of the microcontroller. This storage strategy is employed to optimize resource utilization by enabling the data to be stored permanently without consuming the limited RAM capacity. The dataset used in this study consists of 60 facial images, representing a single facial identity. Each identity is captured at three different distances 10 cm, 20 cm, and 30 cm with five images taken per distance. All data stored in the dataset.cpp file is subsequently uploaded to

the microcontroller and allocated within the RAM nodes located in the RAM discriminators. This structure allows the system to perform facial identification processes efficiently while maintaining optimal memory allocation and minimizing computational overhead.

3.3. Weightless neural network face recognition algorithm

The initial stage of system testing and analysis in this study is the dataset validation process. This step aims to ensure that the facial pattern recognition system functions correctly and is capable of identifying facial data according to the stored identities. Validation is performed by selecting a number of facial pattern samples from the dataset and testing their recognition accuracy. The dataset validation process plays a crucial role in guaranteeing system reliability by ensuring that each facial data entry stored in the database can be accurately recognized based on the individual's identity and image capture distance. In this research, the validated data include three facial classes captured at three different distances: 10 cm, 20 cm, and 30 cm. Each combination of facial class and distance is tested through direct matching against the reference data stored in the dataset, allowing for the evaluation of the system's identification accuracy. The validation results are expected to yield a 100% recognition accuracy, since the test data are directly derived from the same dataset used for training. If the achieved accuracy is under 100%, it indicates that the algorithm has not yet functioned optimally in recognizing its internal facial patterns. The detailed results of the facial dataset validation test are presented in Figure 7.

Figure 7(a) presents the test results for Thread 1, which represents the front-face pattern. Based on the validation results, Face 1 achieved the highest recognition accuracy of 100%, with an execution time of 0.033 seconds. This indicates that the system consistently recognized Face 1 as the correct output. For Face 2, the accuracy reached 45.28% with an execution time of 0.019 seconds, while Face 3 showed an accuracy of 38.14% and an execution time of 0.016 seconds. Meanwhile, the test for Face 4 resulted in an accuracy of 65.81%, with a relatively longer execution time of 0.025 seconds compared to the other faces. A similar pattern can be observed in Figures 7(b) and 7(c), which represent Thread 2 (left-face pattern) and Thread 3 (right-face pattern), respectively. Overall, the validation results show that Face 4 in Thread 3 achieved the highest recognition accuracy of 80.92%, while the fastest execution time was recorded by Thread 1 on Face 3, at 0.016 seconds. Thus, it can be concluded that the dataset recognition validation process operated in accordance with the system's design, where each thread demonstrated a pattern-matching accuracy of up to 100% with respect to its corresponding reference patterns in the dataset.

The next stage involves analyzing the difference in execution time between facial pattern recognition processes that utilize the threading mechanism and those that do not. The non-threaded approach is divided into two methods: i) scanning all facial patterns exhaustively and ii) selecting previously detected facial pattern positions. The first method has a drawback of relatively longer execution time, as the system must recognize patterns sequentially from start to finish, resulting in latency and reduced real-time performance. Conversely, the second method is more time-efficient but risks skipping certain facial pattern groups, potentially lowering system accuracy. In contrast, the use of the thread controller mechanism in this study significantly improves system efficiency. Through this mechanism, multiple facial pattern groups can be executed almost simultaneously, thereby accelerating computation without compromising accuracy. Based on the results presented in Figure 8, the non-threaded approach (a) produced an execution time of 0.271 s, while approach (b) required 0.09 s. The threaded approach also achieved an execution time of 0.09 s. Although the times appear similar, approach (b) actually processes only one-third of the total data handled by the threaded approach. Therefore, the difference in execution time between the two approaches demonstrates a processing efficiency improvement of 0.67%, confirming that the use of a threading mechanism can significantly enhance the performance of the facial pattern recognition system.

After the dataset validation stage was completed, the next step was to test the facial pattern recognition process using the center-scan model approach. In this phase, the system was evaluated by applying the developed method, namely the WNN-FRA [23], [24], which operates through a multithread controller mechanism. This mechanism leverages the processing capabilities of an ARM-based microcontroller, which supports the activation of multiple threads in parallel to improve recognition efficiency. The testing process focused on analyzing the system's performance in recognizing facial patterns from the dataset against primary facial patterns obtained in real time. The test dataset consisted of 60 facial patterns from the first user, while the input data comprised 60 primary facial patterns collected from four facial sample variations, as presented in Table 1. The primary facial patterns served as testing data, with each individual having 15 facial pattern variations. The experiments were conducted using three image-capture distances 10 cm, 20 cm, and 30 cm. Each data capture was performed randomly to assess the system's consistency and reliability under varying conditions. This approach aimed to provide an in-depth evaluation of the system's capability to accurately recognize target faces while considering changes in distance and

facial orientation. The overall results of the facial pattern recognition accuracy using the center-scan scheme are presented in Figure 9.

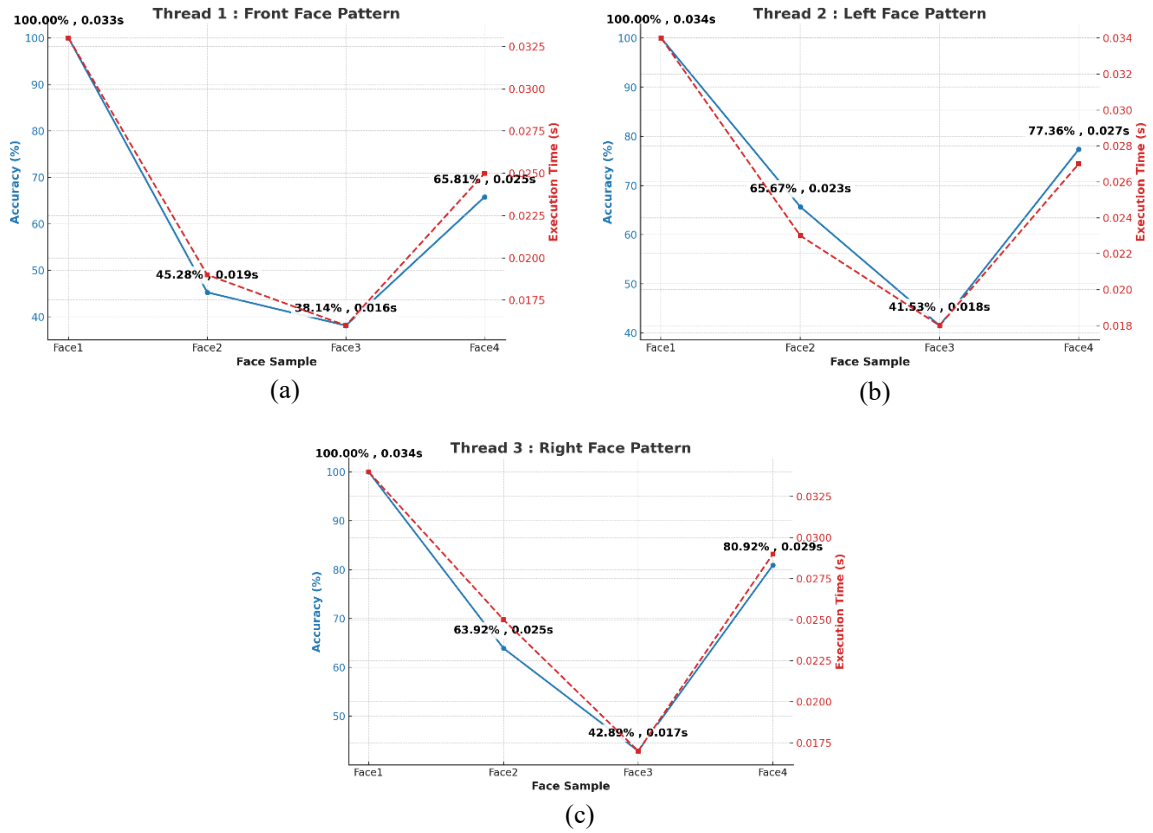


Figure 7. Dataset validation for accuracy and execution time results of (a) Thread 1, (b) Thread 2, and (c) Thread 3

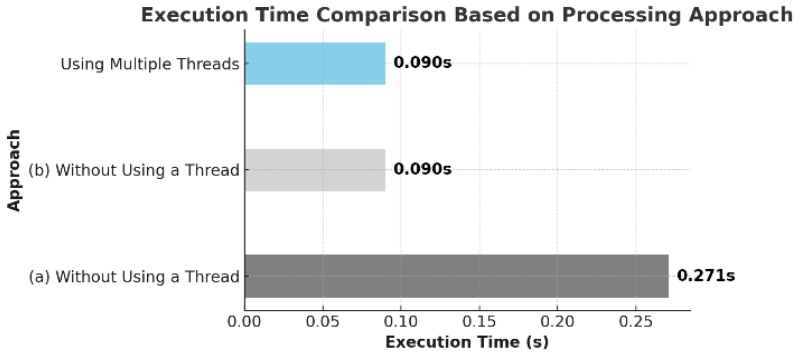


Figure 8. Execution time comparison

Figure 9 presents the results of the accuracy percentage test for facial pattern recognition using the WNN system based on a multi-threading architecture. The horizontal axis (X) represents the dataset index of facial positions ranging from 1 to 60 patterns, while the vertical axis (Y) indicates the recognition accuracy percentage (%). Three color indicators are used to distinguish the testing processes performed in parallel across three threads: Thread 1 for the front-face pattern, Thread 2 for the left-face pattern, and Thread 3 for the right-face pattern. Based on the experimental results, each thread configuration in the multi-threaded system produced distinct accuracy levels for its respective facial pattern group. In Thread 1 (front-face position), the highest accuracy was achieved at pattern index 13, with an accuracy of 94.94%,

representing Face 1. The medium accuracy value was recorded at pattern index 18, with 77.06%, corresponding to Face 3, while the lowest accuracy of 54.03% was observed at pattern index 19, also from Face 3. In Thread 2 (left-face position), the highest accuracy reached 96.47%, obtained at pattern index 25, representing Face 1. The medium accuracy value of 76.47% was recorded at pattern index 40 (Face 4), while the lowest accuracy, 40.86%, occurred at pattern index 35, corresponding to Face 3. Meanwhile, in Thread 3 (right-face position), the best accuracy was achieved at pattern index 53, with a value of 96.83%, representing Face 1. The medium accuracy was observed at pattern index 44, with 76.50% (Face 4), and the lowest accuracy was obtained at pattern index 47, with a value of 41.31%, corresponding to Face 3.

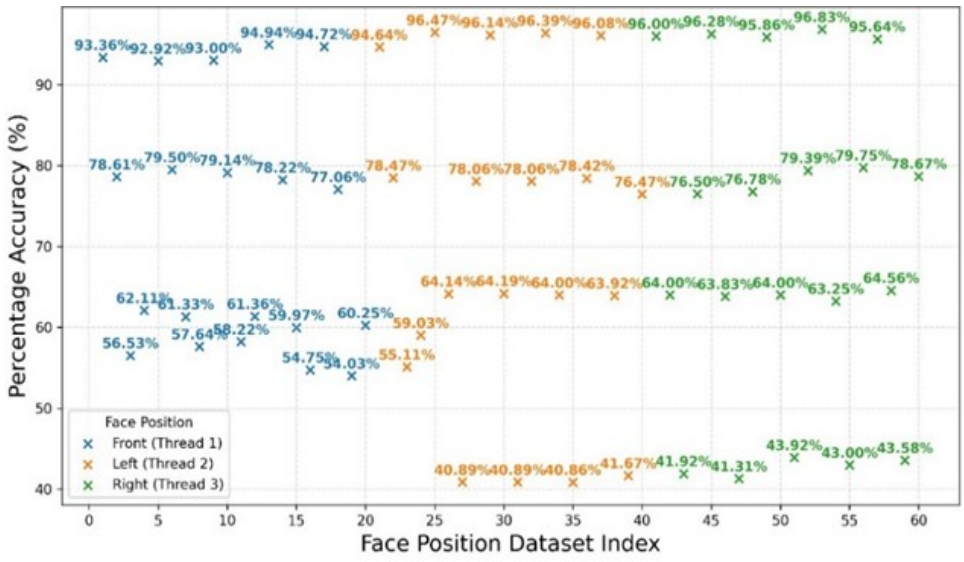


Figure 9. Accuracy test results of the WNN-FRA method

Overall, the experimental results indicate that the highest recognition accuracy was achieved in Thread 3, corresponding to the right-face position. This finding suggests that the WNN-FRA–based system demonstrates greater sensitivity and optimal performance when processing facial patterns from the right-side orientation. This behavior may be attributed to the inherent characteristics of facial features or to a higher degree of correspondence between the training patterns and the input data. The graph also illustrates that all three threads exhibit closely comparable performance, maintaining a high accuracy level (≥90%) at several data points. This observation confirms that the multi-threading mechanism effectively enhances both the efficiency and stability of facial pattern recognition through parallel execution across facial classes. The minor fluctuations observed in the results are primarily influenced by dataset variability and the complexity of facial orientations, rather than by any limitation in the thread architecture itself. Therefore, the implementation of a multi-threaded WNN architecture for facial pattern recognition successfully achieves high accuracy with consistent performance across all dataset groups. These findings reaffirm the effectiveness of the parallel-processing approach in accelerating execution time without compromising classification precision.

Figure 10 illustrates the relationship between the dataset index of facial positions, consisting of 60 facial patterns divided into three threads, each processing 20 facial patterns. The horizontal axis (X) represents the dataset index of facial positions, while the vertical axis (Y) indicates the execution time measured in milliseconds (ms). Each data point corresponds to the duration required by each thread to complete the recognition process for a specific facial pattern. Based on the experimental results, the execution time for all threads falls within the range of 16-34 ms, indicating that the system demonstrates stable performance without any extreme fluctuations. Thread 1, which processes front-face patterns, shows consistent execution stability, with the highest time recorded at 33 ms for pattern index 13 and the lowest at 19 ms for pattern index 19, resulting in a variation of approximately ±14 ms. Similarly, Thread 2 (left-face patterns) exhibits a performance trend comparable to Thread 1, with a maximum execution time of approximately 32 ms at pattern index 26 and a minimum of 16 ms at pattern index 28. This similarity indicates that the computational complexity of recognizing left-face patterns is relatively equivalent to that of front-face patterns. Meanwhile, Thread 3 (right-face patterns) shows the highest execution time of 32 ms at

pattern indices 41, 45, 49, and 57, and the lowest of 16 ms at pattern index 47. The slightly higher fluctuation observed in Thread 3 compared to the other threads is likely due to variations in viewing angles or illumination levels within the right-face dataset.

In addition, an increasing trend in execution time is observed at pattern indices 1, 5, 9, 13, and 17, indicating a periodic tendency. This phenomenon is presumed to be influenced by the complexity of certain facial images, such as variations in rotation, expression, or lighting intensity, which increase the computational load during the classification process. Conversely, lower execution times observed at pattern indices 7, 11, 15, and 19 indicate conditions where the system recognizes patterns more rapidly, suggesting facial features that are easier to distinguish. Overall, the graph demonstrates that the three threads operate within nearly overlapping execution time ranges, indicating the absence of any significant bottlenecks. This finding suggests that the workload distribution mechanism among threads functions in a balanced manner and that the system successfully maintains process synchronization, thereby supporting real-time performance. The average execution time per dataset, ranging from 22-26 ms, is considered fast for an embedded system based on the WNN architecture. Nevertheless, Thread 3 exhibits slightly higher fluctuations, indicating the need for further refinement in the preprocessing or feature normalization stages to enhance the stability of right face recognition. In general, these results confirm that the implementation of the multithreading mechanism is effective in accelerating execution time and improving the overall efficiency of the facial pattern recognition system.

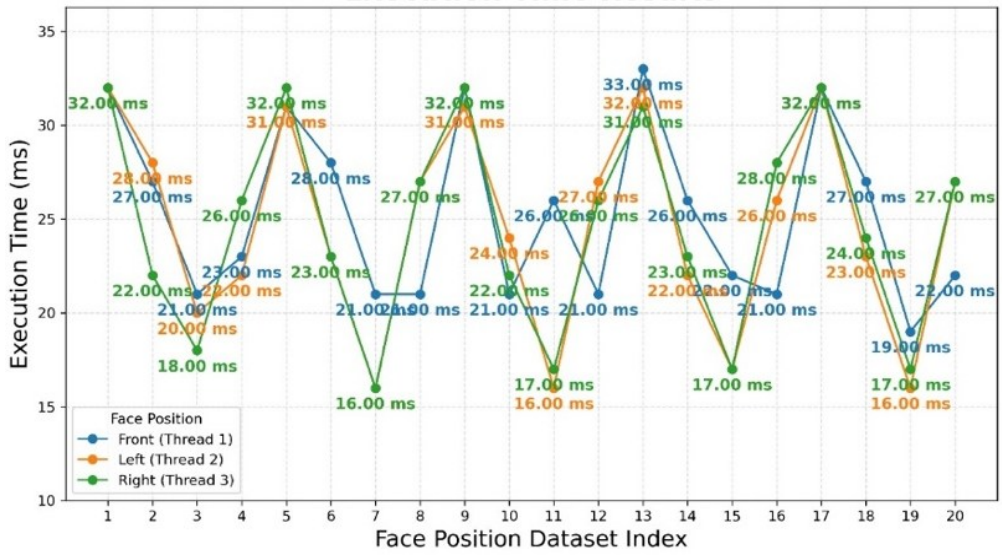


Figure 10. Execution time test results of the WNN-FRA method

4. CONCLUSION

Based on the findings, data analysis, and system performance evaluation, it can be concluded that the implementation of the WNN-FRA integrated with a multi-thread controller has significantly improved both the efficiency and accuracy of facial pattern recognition processes on embedded system platforms. In the conventional single thread configuration, the system required 0.271 seconds to complete the recognition process, whereas the proposed multi-threaded controller achieved an average execution time of only 0.09 seconds, representing an efficiency gain of more than 67%. The workload distribution among threads exhibited a well-balanced performance, with all threads operating consistently within the execution time range of 16-34 milliseconds. The recognition experiments, which involved 60 facial patterns divided into three parallel threads corresponding to front, left, and right facial positions, yielded a maximum accuracy of 96.83% on Thread 3 (right face), with the average overall accuracy exceeding 90% across multiple test points. Memory utilization analysis revealed that the system required only 24 KB of memory, distributed as 9 KB for front-face data, 7.5 KB for left-face data, and 7.5 KB for right-face data, demonstrating exceptional efficiency for an embedded environment with limited hardware resources. The algorithm developed in this study is not yet fully capable of performing comprehensive pattern recognition when differences occur between the pixel values of the input pattern and the reference pattern. Therefore, future research will

incorporate a long short-term memory (LSTM) approach into the RAM discriminator architecture to enable more effective recognition of all pixel variations, thereby improving the overall accuracy of the system.

ACKNOWLEDGMENTS

The authors would like to thank the Faculty of Computer Science, Strategic Pervasive Computing and Intelligent Embedded Systems Research Group (SPCIES-RG), and COMNETS RG, as well as the Universitas Sriwijaya Embedded Systems, Control System, and Robotic Laboratory.

FUNDING INFORMATION

The research publication of this article was funded by the Directorate of Research and Community Service, Directorate General of Research and Development Ministry of Higher Education, Science, and Technology. In accordance with the Implementation Contract of the Operational Assistance Program for State Universities Research Program, Fiscal Year 2025, Contract Number: 109/C3/DT.05.00/PL/2025.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Ahmad Zarkasi	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Hadipurnawan Satria		✓	✓	✓	✓			✓		✓				✓
Anggina Primanita		✓		✓	✓					✓	✓		✓	✓
Deris Stiawan		✓								✓				

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

INFORMED CONSENT

Not applicable, as this study did not involve human participants or personally identifiable information.

ETHICAL APPROVAL

Not applicable, as this study did not involve human or animal subjects and used only non-sensitive computational or public data.

DATA AVAILABILITY

Derived data supporting the findings of this study are available from the corresponding author, [AZ], on request.

REFERENCES

[1] M. Wang and W. Deng, "Deep face recognition: a survey," *Neurocomputing*, vol. 429, pp. 215–244, 2021, doi: 10.1016/j.neucom.2020.10.081
[2] F. Marcolin, E. Vezzetti, and M. G. Monaci, "Face perception foundations for pattern recognition algorithms," *Neurocomputing*, vol. 443, pp. 302–319, 2021, doi: 10.1016/j.neucom.2021.02.074.
[3] S. Wan and S. Goudos, "Faster R-CNN for multi-class fruit detection using a robotic vision system," *Computer Networks*, vol. 168, 2020, doi: 10.1016/j.comnet.2019.107036.

[4] G. Cerutti, R. Prasad, and E. Farella, "Convolutional neural network on embedded platform for people presence detection in low resolution thermal images," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2019-May, pp. 7610–7614, 2019, doi: 10.1109/ICASSP.2019.8682998.
[5] W. J. Song, *Hardware accelerator systems for embedded systems*, 1st ed., vol. 122. Elsevier, 2021, doi: 10.1016/bs.adcom.2020.11.004.
[6] Lv, M. Su, and Z. Wang, "Application of face recognition method under deep learning algorithm in embedded systems," *Microprocessors and Microsystems*, 2021, doi: 10.1016/j.micpro.2021.104034.
[7] S. Nurmaini and A. Zarkasi, "Simple pyramid RAM-based neural network architecture for localization of swarm robots," *Journal of Information Processing Systems*, vol. 11, no. 3, pp. 370–388, 2015, doi: 10.3745/JIPS.01.0008.
[8] L. Santiago *et al.*, "Weightless neural networks as memory-segmented Bloom filters," *Neurocomputing*, vol. 416, pp. 292–304, Nov. 2020, doi: 10.1016/j.neucom.2020.01.115.
[9] L. Oneto, K. Bunte, and A. Sperduti, "Advances in artificial neural networks, machine learning and computational intelligence," *Neurocomputing*, vol. 416, pp. 172–176, 2020, doi: 10.1016/j.neucom.2020.03.059.
[10] O. O. Napoli, A. M. de Almeida, E. Borin, and M. Breternitz, "Memory-efficient DRASiW Models," *Neurocomputing*, vol. 610, Dec. 2024, doi: 10.1016/j.neucom.2024.128443.
[11] M. De Gregorio and M. Giordano, "Background estimation by weightless neural networks," *Pattern Recognition Letters*, vol. 96, pp. 1–11, 2017, doi: 10.1016/j.patrec.2017.05.029.
[12] M. De Gregorio and M. Giordano, "CWISARDH^{+}: background detection in RGBD videos by learning of weightless neural networks," in *New Trends in Image Analysis and Processing – ICIAP 2017*, Cham: Springer, 2017, pp. 242–253, doi: 10.1007/978-3-319-70742-6_23.
[13] D. O. Cardoso, F. França and J. Gama, "A bounded neural network for open set recognition," *2015 International Joint Conference on Neural Networks*, 2015, pp. 1–7, doi: 10.1109/IJCNN.2015.7280680.
[14] D. O. Cardoso, F. M. G. França, and J. Gama, "Weightless neural networks for open set recognition," *Machine Learning*, 2017, doi: 10.1007/s10994-017-5646-4.
[15] M. Berger, A. F. De Souza, J. de O. Neto, E. de Aguiar, and T. O.-Santos, "Visual tracking with VG-RAM weightless neural networks," *Neurocomputing*, vol. 183, pp. 90–105, 2016, doi: 10.1016/j.neucom.2015.04.127.
[16] L. A. Q. Villon *et al.*, "A conditional branch predictor based on weightless neural networks," *Neurocomputing*, vol. 555, Oct. 2023, doi: 10.1016/j.neucom.2023.126637.
[17] R. B. Barbosa, D. L. Cadette Dutra, P. M. V. Lima, D. Carvalho, and F. M. G. França, "Updating KernelCanvas for weightless graph classification," *Neurocomputing*, vol. 646, Sep. 2025, doi: 10.1016/j.neucom.2025.130458.
[18] F. C. Pereira and C. E. Pereira, "Embedded image processing systems for automatic recognition of cracks using UAVs," in *IFAC-PapersOnLine*, Jul. 2015, pp. 16–21, doi: 10.1016/j.ifacol.2015.08.101.
[19] R. Udendhran, M. Balamurugan, A. Suresh, and R. Varatharajan, "Enhancing image processing architecture using deep learning for embedded vision systems," *Microprocessors and Microsystems*, vol. 76, Jul. 2020, doi: 10.1016/j.micpro.2020.103094.
[20] X. Lv, M. Su, and Z. Wang, "Application of face recognition method under deep learning algorithm in embedded systems," *Microprocessors and Microsystems*, Jan. 2021, doi: 10.1016/j.micpro.2021.104034.
[21] A. Zarkasi, H. Ubaya, K. Exaudi, and M. Fitriyanto, "The eye and nose identification chip controller-based on robot vision using weightless neural network method," *Computer Engineering and Applications Journal*, vol. 13, no. 3, 2024.
[22] A. Zarkasi, S. Nurmaini, D. Setiawan, B. Y. Suprapto, H. Ubaya, and R. Kurniati, "Performance comparison of feature face detection algorithm on the embedded platform," *Computer Engineering and Applications*, vol. 11, no. 2, 2022, doi: 10.18495/comengapp.v11i2.405.
[23] A. Zarkasi, H. Ubaya, K. Exaudi, and A. H. Duri, "Implementation of fisherface algorithm for eye and mouth recognition in face-tracking mobile robot," *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika*, vol. 10, no. 3, pp. 556–565, Sep. 2024, doi: 10.26555/jiteki.v10i3.29266.
[24] A. Zarkasi, S. Nurmaini, D. Stiawan, and B. Y. Suprapto, "Weightless neural networks face recognition learning process for binary facial pattern," *Indonesian Journal of Electrical Engineering and Informatics*, vol. 10, no. 4, pp. 955–969, Dec. 2022, doi: 10.52549/ijeei.v10.i4.3957.
[25] S. Catalán, J. R. Herrero, F. D. Igual, R. R.-Sánchez, E. S. Q.-Ortí, and C. A.-Jones, "Multi-threaded dense linear algebra libraries for low-power asymmetric multicore processors," *Journal of Computer Science*, vol. 25, pp. 140–151, Mar. 2018, doi: 10.1016/j.jocs.2016.10.020.
[26] J. Lu and C. H. Rycroft, "TRIME++: multi-threaded triangular meshing in two dimensions," *Computer Physics Communications*, vol. 308, Mar. 2025, doi: 10.1016/j.cpc.2024.109442.
[27] J. Feliu, A. Ros, M. E. Acacio, and S. Kaxiras, "Speculative inter-thread store-to-load forwarding in SMT architectures," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 94–106, Mar. 2023, doi: 10.1016/j.jpdc.2022.11.007.
[28] S. Tošić, N. Vojnović, M. Vidaković, J. Vidaković, V. Levi, and D. Četenović, "Single and multi-threaded power flow algorithm for integrated transmission-distribution-residential networks," *Computers and Electrical Engineering*, vol. 120, Dec. 2024, doi: 10.1016/j.compeleceng.2024.109735.
[29] W. Wang and W. M. Lin, "An integrated autonomous control mechanism to optimize sharing of multiple resources in simultaneous multi-threading processors," *Journal of Systems Architecture*, vol. 88, pp. 87–96, Aug. 2018, doi: 10.1016/j.sysarc.2018.06.003.
[30] D. C.-Caballero, F. D.-del-Rio, D. C.-Muñiz, D. O.-Martín, and I. P.-Hurtado, "A new approach for software-simulation of membrane systems using a multi-thread programming model," *Simulation Modelling Practice and Theory*, vol. 136, Nov. 2024, doi: 10.1016/j.simpat.2024.103007.
[31] S. H. Lee, "Real-time edge computing on multi-processes and multi-threading architectures for deep learning applications," *Microprocess Microsyst*, vol. 92, Jul. 2022, doi: 10.1016/j.micpro.2022.104554.
[32] J. Zhao, Y. Fu, Y. Wu, J. Dong, and R. Hong, "Thread-sensitive fuzzing for concurrency bug detection," *Computers & Security*, vol. 148, Jan. 2025, doi: 10.1016/j.cose.2024.104171.
[33] M. Samadi, S. Royuela, L. M. Pinho, T. Carvalho, and E. Quiñones, "Time-predictable task-to-thread mapping in multi-core processors," *Journal of Systems Architecture*, vol. 148, Mar. 2024, doi: 10.1016/j.sysarc.2024.103068.
[34] V. Kelefouras and K. Djemame, "Workflow simulation and multi-threading aware task scheduling for heterogeneous computing," *Journal of Parallel and Distributed Computing*, vol. 168, pp. 17–32, Oct. 2022, doi: 10.1016/j.jpdc.2022.05.011.
[35] M. Rezazadeh, N. E.-Jivan, S. V. Azhari, and M. R. Dagenais, "Performance evaluation of complex multi-thread applications through execution path analysis," *Performance Evaluation*, vol. 155–156, Jun. 2022, doi: 10.1016/j.peva.2022.102289.

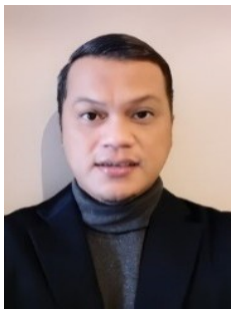
[36] T. Wu, J. He, Y. Qian, and W. Liu, "Optimizing the performance of in-memory file system by thread scheduling and file migration under NUMA multiprocessor systems," *Journal of Systems Architecture*, vol. 159, Feb. 2025, doi: 10.1016/j.sysarc.2025.103344.
[37] Y. Zhang, Z. Song, J. Yu, B. Cao, and L. Wang, "A novel pose estimation method for robot threaded assembly pre-alignment based on binocular vision," *Robotics and Computer-Integrated Manufacturing*, vol. 93, Jun. 2025, doi: 10.1016/j.rcim.2024.102939.

BIOGRAPHIES OF AUTHORS

Ahmad Zarkasi was born in Palembang on August 25, 1979. In 2023, he obtained his doctor in Informatics Engineering from the Universitas Sriwijaya, Faculty of Engineering. His research interests include microprocessors, system-on-chip (SoC), embedded systems, and robotics. His research interests include subjects such as WNNs in robotic systems and pattern recognition for robot vision. Currently, he is a lecturer in the Department of Computer Engineering, Faculty of Computer Science, Universitas Sriwijaya, Indonesia. He can be contacted at email: zarkasi98@gmail.com.

Hadipurnawan Satria was born in Palembang on April 18, 1980. Currently, he is a lecturer in the Department of Informatics Engineering, Faculty of Computer Science, Universitas Sriwijaya, Indonesia. He received her Ph.D. in Computer Science from Sun Moon University's Faculty of Computer Science in 2010. His research interests include a virtual development environment for embedded software (VDEES). He can be contacted at email: hadipurnawan.satria@gmail.com.

Anggina Primanita was born in Bekasi on August 6, 1989. Currently, she is a lecturer in the Department of Informatics Engineering, Faculty of Computer Science, Universitas Sriwijaya, Indonesia. She earned a Ph.D. from the Japan Advanced Institute of Science and Technology at the Faculty of Information Science in 2021. Her research interests are in computing games. She can be contacted at email: anggina.primanita@gmail.com.



Deris Stiawan received the Ph.D. degree in Computer Science from the Universiti Teknologi Malaysia, Malaysia. He is currently a professor in the Department of Computer Engineering, Faculty of Computer Science, Universitas Sriwijaya. His research interests include computer networks, intrusion detection/prevention systems, and heterogeneous networks. He can be contacted at email: deris@unsri.ac.id.