International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-3, February 2020

Implementation of Text Compression using Adaptive Shannon-Fano Algorithm

Satria Gunawan Zain, Nirwana, Andi Baso Kaswar, Suhartono, Abd. Rahman Patta

Abstract: ***This study aims to implement the Shannon-fano Adaptive data compression algorithm on characters as input data. This study also investigates the data compression ratio, which is the ratio between the number of data bits before and after compression. The resulting program is tested by using black-box testing, measuring the number of character variants and the number of types of characters to the compression ratio, and testing the objective truth with the Mean Square Error (MSE) method. The description of the characteristics of the application made is done by processing data in the form of a collection of characters that have different types of characters, variants, and the number of characters. This research presents algorithm that support the steps of making adaptive Shannon-fano compression applications. The length of the character determines the variant value, compression ratio, and the number of input character types. Based on the results of test results, no error occurs according to the comparison of the original text input and the decompression results. A higher appearance frequency of a character causes a greater compression ratio of the resulting file; the analysis shows that a higher number of types of input characters causes a lower compression ratio, which proves that the proposed method in real-time data compression improves the effectiveness and efficiency of the compression process.***

Keywords**:** ***Data Compression, Shannon-Fano, Text Compression***

I. INTRODUCTION

Most of the data management has been performed routinely using computers. Along with developments in the field of telecommunications, the amount of information collected, processed, and then prepared to be accessed via the internet is increasing significantly. Recent advances in information technology are causing massive amounts of data generated every second. Consequently, data storage and transmission tend to increase to an extraordinary level [1]. The demand for data exchange between users and the number of users causes the increasing necessity for data exchange channels (bandwidth) and data storage media. Investment to satisfy those needs is not meager. One of the efforts to reduce the cost of providing infrastructure so that services that can satisfy user needs can be provided is to reduce the amount of data available on the communication channel without limiting the flow of data exchange between users. The exchanged data or files have many types, including image or image, text, compressed application, etc. Even large data storage on the server is often compressed. Currently, data compression is one of the subjects in information technology that is currently widely applied. Data compression techniques positively contribute to saving the use of communication channels and file storage media. The file size also affects the speed in data exchange between users. A preprocessing method for universal text compression has been developed [2]. The method integrates five algorithms, including capital letter conversion, end of line (EOL) coding, word replacement, phrase replacement, and letter recording. This method does not depend on the type of language or dictionary. However, this method requires high costs in preprocessing. In other studies, [3] proposed a method that presents a more efficient technique called the b64 package technique for short messages. The proposed algorithm is efficient, lightweight, and easily operated. This method is more efficient than compress, gzip, and bzip2 methods. It has a b64 package that operates in two phases. Next, [4] proposed a greedy approach to static text compression. The proposed method utilizes a finite state machine from a greedy-based compression method with an arbitrary dictionary to obtain a fast distribution system. [5] uses a graphics-based method to obtain the sequence of characters to be processed in the compression process. This proposed method is to build a graph in one text delivery and then mine all the determined patterns to be compressed in one graph trajectory. Then, [6] introduces a new compression technique, Novel FIM based Huffman coding techniques using hash tables (FPH2) for text compression in the process of calculating frequently occurring patterns. [7] offers a compression algorithm based on displaying original data. Represented in bits, to the surface of the plane with subsequent searches for the same region. This method shows that the proposed method can provide a high level of compression from various types of telemetry data. The necessity for data compression is not only marked from the size reduction, but the speed of compression is also a parameter in assessing the efficiency of the techniques used.

Several techniques do not require data completeness to begin the compression process, such as the Lampel Ziv Welch (LZW) method [8], which is dictionary-based and statistical-based Huffman compression.

Revised Manuscript Received on February 05, 2020.

* Correspondence Author

Satria Gunawan Zain*, Computer Engineering, Universitas Negeri Makassar, Makassar, Indonesia. Email: satria.gunawan.zain@unm.ac.id

Nirwana, Computer and Informatics Engineering Education, Universitas Negeri Makassar, Makassar, Indonesia. Email: Nirwana_Ptik@unm.ac.id

Andi Baso Kaswar, Computer Engineering, Universitas Negeri Makassar, Makassar, Indonesia. Email: a.baso.kaswar@unm.ac.id

Suhartono, Computer Engineering, Universitas Negeri Makassar, Makassar. Indonesia. Email : suhartono@unm.ac.id

Abd. Rahman Patta, Computer Engineering, Universitas Negeri Makassar, Makassar, Indonesia. Email: abd.rahman.patta@unm.ac.id

The standard compression technique still uses non-adaptive compression techniques that require the completeness of data files for compression.

This study will examine the implementation of the adaptive Shannon-fano compression algorithm and utilize Matlab to build the steps of implementing the adaptive Shannon-Fano algorithm. The implementation steps of the algorithm built to provide an overview of the implementation of the adaptive Shannon-fano algorithm in embedded microcontroller systems to support more efficient data communication, which is even possible to be used in completing data communication protocols for channel efficiency and data exchange speed.

The problems above designate that the data compression process that uses the Shannon-Fano Adaptive process works in real-time with a direct compression process; if there is a data read, then it is directly compressed, in contrast with Non-Adaptive Shannon-Fano, that wait for the input data to complete before the compression process.

II. METHOD

A. Design Model

The Shannon-Fano algorithm is a method that was first known to be able to encode symbols effectively, one algorithm that can compress data very well without losing lost bits. This method was developed together by Claude Shannon at Bell Laboratory and Robert Fano at MIT in 1949. Shannon and Fano developed algorithmic coding techniques based on the variable-length code that some characters in the data to be encoded are represented by codes that are shorter than the characters in the data. The lower appearance frequency of characters results in the longer code. Hence, the resulting code is varied in length and unique. In principle, this algorithm uses a top-down approach in the preparation of binary trees.

In data compression, Shannon-Fano coding is a technique to form a prefix code based on a set of symbols and their probabilities.

Fig. 1 and Fig. 2 show The process of compression and decompression. Variable X_0 and X_1 represented compression data and compressed data, respectively. The compressed data flowchart and decompression data flowchart can be seen in Fig. 3 and Fig. 4.

According to the flowchart in Figure 3, the steps for real-time data compression are as follows:

1. Initialize the streaming data to be processed
2. Form arrays and statistical code tables
3. Read the streaming data input character. If there is an input

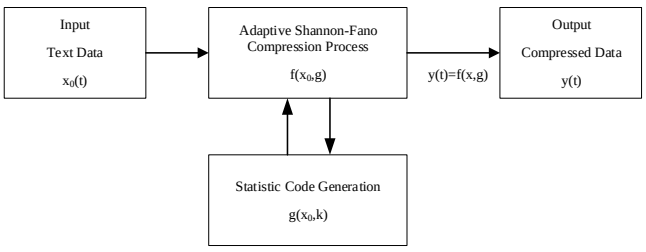

Fig. 1. Design Model of Adaptive Shannon - Fano Compression

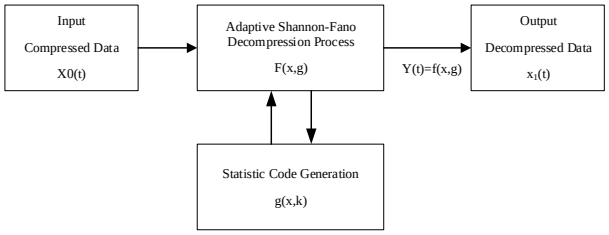

Fig. 2. Design Model of Adaptive Shannon - Fano Decompression

character(s), proceed to step four. Otherwise, stop the process.

4. Check if the character(s) is already in the statistics table, remove the code output for the character and return to step three.

   Otherwise, do the following steps:

   a. Generate ESC code and ASCII code from the character
   b. Add ESC weights to the table
   c. Add the C character to the statistics table and its weight
   d. Calculate new code from a statistic array
   e. Return to step three

5. Repeat the step three.

In the real-time data decompression flowchart, the steps are as follows:

1. Initialize the data to be processed
2. Form arrays and statistic code
3. Read the character input stream, if there is no character input, then stop the process. Otherwise continue to step 4
4. Check whether K is the same as escape code
5. If K is not the same as escape code, proceed to step six. If K is the same as escape code, do the following steps:
   a. Read the ASCII C character output from the input stream
   b. Add an escape button to the statistics
   c. Add character C to the statistical table with a weight of 1
   d. Return to step three
6. Check the output of letter C, which has the word K
7. Add the weight of the letter C
8. Calculate the new code from the Statistics Array

B. Example of Adaptive Shannon-Fano Algorithm Implementation

Table I explains that at the beginning before the incoming text message, the table is still considered as an empty table so that the escaped code is zero.

Then the compression algorithm reads the first letter of the message, which is "g"; since "g" is not yet in the statistics table, "g" cannot be directly encoded, it must have output code for escape plus the ASCII code (in 8 bits) for the letter "g." Thus the output for the letter "g" becomes "001100111". After encoding, the statistics change by adding one cardinality to escape and adding the letter "g" to the table with cardinality is equal to one.

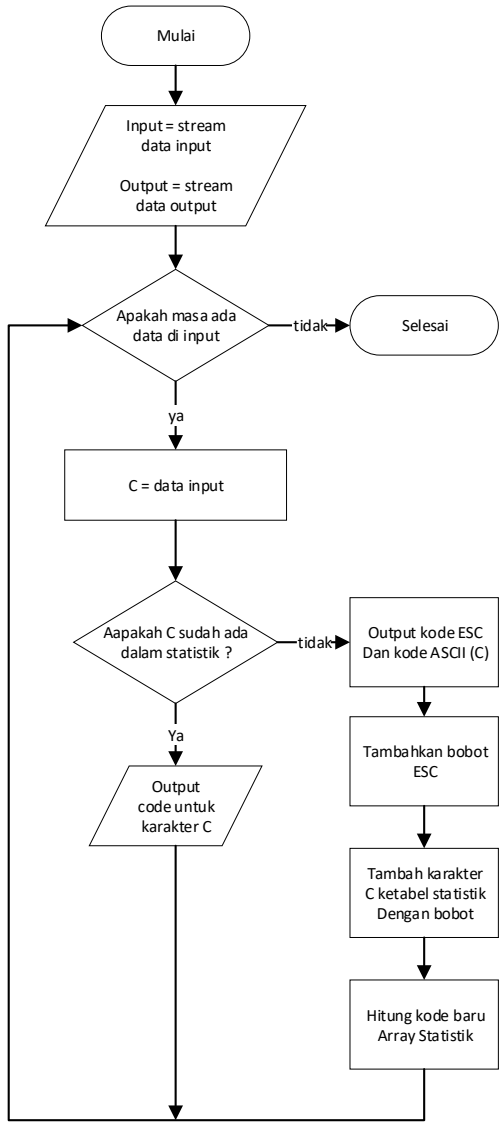

Fig. 3. Adaptive Shannon - Fano Compression Flowchart

The same condition is applicable for input "a"; the output becomes the code fc Input :ape sign along with the ASCII code for "a" in 8 bits (the result is "001100001"). After adding the cardinality of escape and entering "a" into the table, then recalculate the code of each letter, as shown in Table III. As in the previous step or table, in Table X, incoming letters are sorted by ASCII code and then added by zero, and the next line added by one; the escaped value is added by one to be equal to three. Changes to the code are adjusted based on binary logic.VZ

The ASCII code of letter j is smaller th... the previous letters; hence, in Table VI, the letter j remains on the last line, the escape value is added by one becomes equal to four. For four input data coding, the code is adjusted by binary logic using two bits in the letter a-d and the next two bits in the letter g-j.

In contrast to the previous step, in Table VI, the letter appears for the second time so that the value of the number of escapes remains, and the number of probabilities is added by one becomes equal to two.

The next letter that appears is the letter h, so the processing steps in Table VII are the same as in Tables I to Table V, with no change in the number of probabilities. Likewise, the next letter in Table VIII that appears is the letter m with one probability; therefore, the code change process is the same as in Tables I to Table V.

The letters that appear in Table IX are the letters "a"; hence the escape value is fixed, and the number of probabilities is added by one and equals three.

Moreover, the letter d appears for the second time Table X; hence the process in Table VI and Table IX is applicable by adding the probability number letter d to be equal to two, and no change in the escape value.

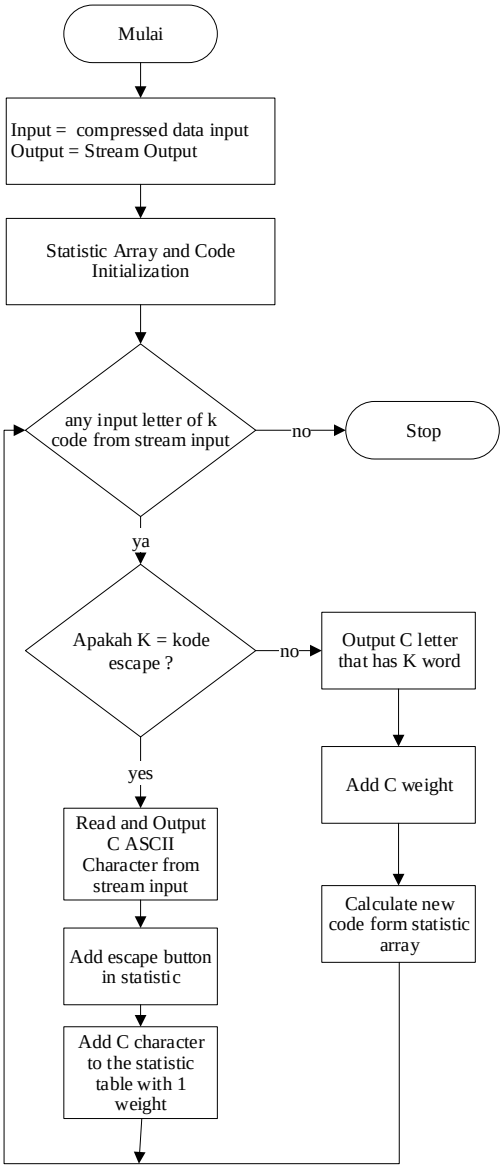

Fig. 4. Adaptive Shannon - Fano Deompression Flowchart

Finish
no
yes
Is C already in the statistics?
Output ESC code and ASCII code (C)

Table- III: Second Letter Data (a)

Letter	Count	Code
Escape	2	0
A	1	10
G	1	11

Table- IV: Third Letter Data (d)

Letter	Count	Code
Escape	3	0
A	1	10
D	1	110
G	1	111

Table- V: Fourth Letter Data (j)

Letter	Count	Code
Escape	3	0
A	1	100
D	1	101
G	1	110
J	1	111

Table- VI: Fifth Letter Data (a)

Letter	Count	Code
Escape	4	0
A	2	100
D	1	101
G	1	1110
J	1	1111

Table- VIII: Seventh Letter Data (m)

Letter	Count	Code
Escape	6	0
A	2	100
D	1	101
G	1	1100
J	1	1110
M	1	1111

Table- IX: Eighth Letter Data (a)

Letter	Count	Code
Escape	6	0
A	3	100
D	1	101
G	1	1100
H	1	1101
J	1	1110
M	1	1111

Table- X: Ninth Letter Data (d)

Letter	Count	Code
Escape	6	0
A	3	100
D	1	101
G	1	1100
H	1	1101
J	1	1110
M	1	1111

Table- XI: Tenth Letter Data (a)

Letter	Count	Code
Escape	6	0
A	4	10
D	2	1100
G	1	1101
H	1	1110
J	1	11110
M	1	11111

For the last data, the letter that appears is "a" for the fourth time, so Table XI above is a complete table for one text message "gadjahMada". The Overall coding of the letters of the text message "gadjahMada" shown in (Table 2.18); therefore, the text message has 34 bits in total.

III. RESULT AND DISCUSSION

This section explains the research result. The compression process of 7 characters using the Adaptive Shannon-Fano Algorithm results in a compression ratio of 1.09804 and data redundancy of around 0.0892857. Comparison of MSE (mean square error) is obtained by comparing the size of data capacity and the number of characters, of the original file and the file after decompression. However, the is no error found in the file compression process using the Adaptive Shannon-Fano Algorithm.

The process works in real-time and uses a lassless compression technique, although this compression technique has a lower degree of compression but data accuracy is maintained before and after the compression process. In this last less compression, missing bits from the data compression process is not permitted.

The Variance Value of the average code is obtained by calculating its value in the program using the syntax “var (variable) enter” for example, the variable “a = ‘assalamualaikum’ enter” then the value “var (a) = 59.6667”. The average code data is the number of bits in the decompressed file divided by the number of characters of the original file.

The compression system used in this study is inputting the original file, which input per characters, and a compression process that occurs for each character input.

Table- XII: Compression Result of Dynamic Method

Letter	Count	Bit
G	4	1101
a	2	10
D	4	1100
j	5	11110
a	2	10
H	4	1110
M	5	111111
A	2	10
d	4	1100
A	2	10

Characters from the original data then convert into ASCII code, ASCII code of each character of input data is also converted to binary data. After the conversion process of characters into ASCII code and binary code, then the number of data bits and input data characters are calculated. Furthermore, the original input data file then compressed. The file compression process works in real-time using the Adaptive Shannon - Fano Algorithm, where every character of input data is directly compressed. At the original file compressing step, the output file compression code is displayed first.

The output file compression is obtained from the original input character file, each character input during the compression process through repeated data checks, for example, the first input character 'a' the input data is stored in the statistics table, and the code is ASCII code. The input of the second characters 'b' is also stored in a statistical table, and the code is ASCII code, this process is repeated for every character input. After getting the same character input, then check the statistics table.

If the input character is already in the statistic table, then it is immediately encoded in the statistic table, the output code is a code from the statistic table. Input data is then stored in a table, each input data is coded into ASCII code, with "cek_tabel" syntax in the program. The code in the statistical table is determined by dividing the two groups of data input characters, where the division of the above characters and the below characters have a value of '1' and '0', respectively. The output code uses the existed code if it is already in the statistic table. After knowing the code of the input data, the statistic table is updated by using the "update_tabel" function in the syntax.

In the Shannon-Fano Adaptive application, the original file is inputted and compressed in real-time per character. After the compression process, the compressed file is then decompressed to examine the data authenticity obtained from the compression process. In the decompressing process, the ASCII compressed file is compressed and converted to characters that correspond with the input characters. The decompressing process is real-time using Adaptive Shannon – Fano Algorithm, each character input of the original file is immediately compressed and read in decompression. This process continues during the input of characters of the original file.

After displaying the decompression results, the decompression results are converted to binary code in order to facilitate the determination of the number of bits and the number of characters decompressed to investigate whether it corresponds to the original file or not and the results shows that there is no different in the original, compressed, and decompressed file, the number of bits and the number of characters for each process remain the same.

In this research, the decompression process can be determined by converting the ASCII code data back into compressed binary code, then counting the number of data compression bits and then dividing it into 8 bits and compressed per 8 bits for one character. In ASCII to binary compression, there is an intermediate code where the code in the form of '0' will not be encoded; hence, in the syntax, if the binary data k is equal to k + 1, then the out is equal to '0'. The compression output process starts from k + 2 to k + 9, where data conversion of binary to decimal to obtain output continues to repeat for every input character with the "bin2dec" data conversion code.

Table- XIII: Compression Test with Variant Data

No	Real-time Data Input	Length of Input Data	Variant	Compression Ratio	Average Code Length
1	assalamualaikum	15	1.379310345	0.725	5.8
2	Kompresi fileaa	15	1.015873016	0.984375	7.875
3	tugas akhir ana	15	1.066666667	0.9375	7.5
4	programkompresi	15	1.091405184	0.91625	7.33
5	sistemdekompres	15	1.121547736	0.891625	7.133
6	Dekompresi data	15	0.983646871	1.016625	8.133
7	nilairasio text	15	1.061993894	0.941625	7.533
8	fakultas teknik	15	1.03452735	0.966625	7.733
9	tetap jaya unm1	15	1.071524243	0.93325	7.466
10	lab pte ft unm1	15	1.008445733	0.991625	7.933

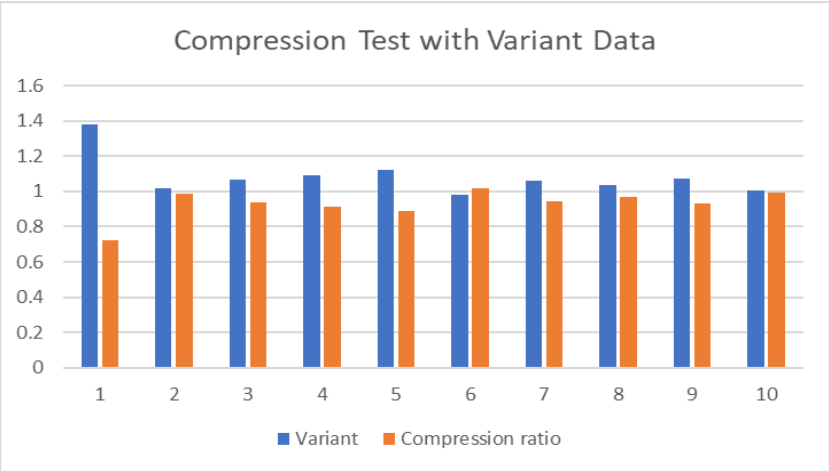

Fig. 5. Comparisson of variant and compressiion ratio result

Table XIII and Fig. 5 show that the compression of a data also affected by the variety of characters that are collected; a higher appearance frequency of an input data character causes a higher compression ratio while the more types of characters inputted cause the lower data compression ratio. These data shows that the proposed method can provide a high compression ratio.

For example, we input real-time data "assalamualaikum" with 120 number of bits. Then the ASCII code will be generated "97 115 115 97 108 97 109 117 97 108 97 105 107 117 109" and binary code "110000111100111110011110000111011001100000111011101111010111000011101100110000111010011101110101010100101010101010101". The output of the string compression file is "097011511110108100109011710111010010501071111111111" with87 number of bits. From this string, the MSE is 0, and the Compression Ratio is 1.37931, which shows that the proposed method provides good compression results in real-time and produce the right decompression results without any slightest error.

IV. CONCLUSION

The research concludes that the method implemented in the microcontroller embedded system generates a higher ratio of data compression using the adaptive Shannon-fano algorithm, which defined by the comparison of compression ratios after variant data values determined and the number of characters types inputted.

The data compression process using the Adaptive Shannon-fano algorithm is more effective than the non-adaptive Shannon-fano algorithm, which attested by Mean Square Error (MSE) of data compression error. Compression using the non-adaptive Shannon-fano algorithm in the MSE test contained an error after the data decompression process. In contrast, data compression using the adaptive Shannon-fano algorithm results in no error between the original file and the file after decompression, which is a perfect solution in the business and educational world because this compression is suitable for fast or real-time file transfers without reducing data quality.

Moreover, the result concludes that the compression ratio for higher character variants has a lower ratio value. The compression ratio is inversely proportional to the character variant of the data stream.

Adaptive Shannon-fano compression can be developed for any application, such as the development of file transfer processes via protocols using offline network systems such as ad hoc and online file exchanges.

REFERENCES

1. J. Uthayakumar, T. Vengattaraman, and P. Dhavachelvan, "A Survey on Data Compression Techniques: From the Perspective of Data Quality, Coding Schemes, Data Type and Applications," J. King Saud Univ. - Comput. Inf. Sci., 2018.
2. J. Abel and W. Teahan, "Universal text preprocessing for data compression," IEEE Trans. Comput., vol. 54, no. 5, pp. 497–507, 2005.
3. K. Kalajdzic, S. H. Ali, and A. Patel, "Rapid lossless compression of short text messages," Comput. Stand. Interfaces, vol. 37, pp. 53–59, 2015.
4. S. De Agostino, "The greedy approach to dictionary-based static text compression on a distributed system," J. Discret. Algorithms, vol. 34, pp. 54–61, 2015.
5. C. Oswald, A. I. Ghosh, and B. Sivaselvan, "Knowledge engineering perspective of text compression," in 2015 Annual IEEE India Conference (INDICON), 2015, pp. 1–6.
6. C. Oswald and B. Sivaselvan, "An optimal text compression algorithm based on frequent pattern mining," J. Ambient Intell. Humaniz. Comput., vol. 9, no. 3, pp. 803–822, 2018.
7. A. V. Levenets, I. V. Bogachev, and E. U. Chye, "Telemetry Data Compression Algorithms Based On Operation of Displaying onto Geometric Surfaces," in 2017 International Siberian Conference on Control and Communications, SIBCON 2017, 2017, pp. 1–6.
8. F. Suhastra, "Implementasi Algoritma Kompresi Lampel Ziv Welch (LZW) Pada Berkas Digital," Pelita Manaj. Budi Darma, pp. 54–57, 2014.

AUTHORS PROFILE

Satria Gunawan Zain completed his bachelor's degree in Electronics Engineering Education at Universitas Negeri Makassar in 2005, Masters degree in Electrical Engineering at Universitas Gadjahmada in 2007, and Doctoral in Electrical Engineering at Universitas Gadjahmada in 2014. The focus research is signal processing. Email: satria.gunawan.zain@unm.ac.id

Nirwana completed her bachelor degree at Universitas Negeri Makassar. The focus of her research was on signal processing. Email: Nirwana_Ptik@unm.ac.id

Andi Baso Kaswar completed his bachelor's degree at Universitas Negeri Makassar and master's degree at Institut Teknologi Sepuluh Nopember Surabaya. The focus of his research is digital image processing and computer vision. currently active as a lecturer at Universitas Negeri Makassar. Email: a.baso.kaswar@unm.ac.id.

Suhartono. Computer Engineering Study Program, Computer Science, Universitas Negeri Makassar, Indonesia. Research field area are IoT, AI, Smart Device, Database Security, Information System.

.

Abd. Rahman Patta, complated his Master's Degree at Hasanuddin University. The focus Research Information System, Information Security, AI
abd.rahman.patta@unm.ac.id