



PRACTICAL WORK – IF184801

Implementation and Evaluation of FuzzyStego Steganography Scheme in Net Centric Computing Lab

Department of Informatics ITS
Keputih, Sukolilo, Surabaya, East Java 60117
Period: July 2024 – December 2024

By:

Mardhatillah Shevy Ananti

5025211070

Department Advisor
Hudan Studiawan, S.Kom., M.Kom., Ph.D.
Field Supervisor
Prof. Tohari Ahmad, S.Kom., MIT., Ph.D.

DEPARTMENT OF INFORMATICS
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2025

[Halaman ini sengaja dikosongkan]

LIST OF CONTENTS

LIST OF CONTENTS.....	iii
LIST OF FIGURES.....	vii
LIST OF TABLES	ix
VALIDITY SHEET	xi
FOREWORD.....	xv
CHAPTER 1 INTRODUCTION.....	1
1.1. Background.....	1
1.2. Objective.....	1
1.3. Benefit.....	2
1.4. Formulation of the problem	2
1.5. Location and Time of Internship.....	2
1.6. Practical Work Methodology	3
1.6.1. Literature Study	3
1.6.2. Algorithm Implementation	3
1.6.3. Performance Testing.....	4
1.6.4. Results Analysis	4
1.7. Report Systematics.....	5
1.7.1. Chapter I Introduction	5
1.7.2. Chapter II Company Profile.....	5
1.7.3. Chapter III Literature Review.....	5
1.7.4. Chapter IV System Implementation	5
1.7.5. Chapter V Testing and Analysis of Results	5

1.7.6. Chapter VI Conclusions and Suggestions.....	5
CHAPTER 2 LABORATORY PROFILE	7
2.1. Net Centric Computing Laboratory Profile.....	7
2.2. Location	7
CHAPTER 3 LITERATURE REVIEW.....	9
3.1. Steganography.....	9
3.2. Fuzzy Logic in Steganography	9
3.3. Relevant Steganography Methods	10
3.3.1. Least Significant Bit (LSB)	10
3.3.2. Pixel Value Differentiation (PVD)	10
3.3.3. Transform Domain Techniques	11
3.4. Steganography Quality Evaluation Parameters.....	11
3.4.1. Peak Signal-to-Noise Ratio (PSNR).....	11
3.4.2. Structural Similarity Index Measure (SSIM).....	12
CHAPTER 4 SYSTEM IMPLEMENTATION	15
4.1. Data Preparation.....	15
4.2. Implementation of FuzzyStego Algorithm.....	15
4.2.1. Embedding Algorithm	16
4.2.2. Extracting Algorithm.....	17
4.3. Performance Testing and Evaluation	18
4.3.1. Testing Methods	18
4.3.2. Pseudocodes of Matlab Functions	18
CHAPTER 5 TESTING AND ANALYSIS OF RESULTS	26

5.1.	Experimental Results	26
5.2.	Comparative Analysis with Previous Methods	28
CHAPTER 6 CONCLUSION AND SUGGESTIONS		35
7.1.	Conclusion	35
7.2.	Suggestion.....	36
BIBLIOGRAPHY		38
ATTACHMENT		42
AUTHOR BIOGRAPHY		55

[Halaman ini sengaja dikosongkan]

LIST OF FIGURES

Figure 1. Methodology Flowchart of FuzzyStego Method	16
Figure 2. PSNR Results of FuzzyStego and Existing Algorithms	29
Figure 3. EC Results of FuzzyStego and Existing Algorithms ...	30
Figure 4. Cover and Stego Images using FuzzyStego Histogram	31
Figure 5. Ablation Experiments Results through PSNR Results.	33
Figure 6. Ablation Experiments Results through SSIM Results .	33
Figure 7. Embedding and Calculating PSNR	45
Figure 8. Extracting and Calculating PSNR.....	49
Figure 9. Compression and Calculating PSNR	52
Figure 10. No Fuzzy Logic and Calculating Evaluation Metrics	54

[Halaman ini sengaja dikosongkan]

LIST OF TABLES

Table 1. PSNR and SSIM Results of FuzzyStego	26
Table 2. Huffman Compression PSNR and SSIM Results.....	27

[Halaman ini sengaja dikosongkan]

**VALIDITY SHEET
PRACTICAL WORK**

**Implementation and Evaluation of FuzzyStego
Steganography Scheme in Net Centric Computing Lab**

By:

Mardhatillah Shevy Ananti

5025211070

Approved by the Internship Supervisor:

1. Hudan Studiawan, S.Kom.,
M.Kom., Ph.D.
NIP.



(Department Supervisor)

2. Prof. Tohari Ahmad, S.Kom.,
MIT., Ph.D.
NIP.



(Field Supervisor)

[Halaman ini sengaja dikosongkan]

Implementation and Evaluation of FuzzyStego Steganography Scheme in Net Centric Computing Lab

Student Name : Mardhatillah Shevy Ananti
NRP : 5025211070
Department : FTEIC-ITS Informatics
Department Supervisor : Hudan Studiawan, S.Kom., M.Kom.,
Ph.D.
Field Supervisor : Prof. Tohari Ahmad, S.Kom.,
MIT.,Ph.D.

ABSTRACT

Steganography is a technique in Information Security that works by hiding confidential data in digital media without causing any noticeable changes, this include images, audios or text. One of the main challenges in steganography is to keep a balance between data embedding capacity and quality of the embedded image (stego image). In this study, the focus will be on the development and evaluation of an image steganography method called FuzzyStego. An adaptive steganography scheme based on fuzzy logic, this method aims to optimize the data embedding area in the spatial domain of images.

To help the implementation of the FuzzyStego algorithm, the whole process was carried out in the Net Centric Computing Lab, applying a fuzzy logic approach that classifies pixels in the image into five intensity levels: Low (L), Medium-Low (ML), Medium (M), Medium-High (MH), and High (H). Based on these intensity levels, the number of bits that can be inserted in each pixel is determined adaptively to minimize visual distortion.

Keywords: Steganography, fuzzy logic, information security, data hiding, image quality

[Halaman ini sengaja dikosongkan]

FOREWORD

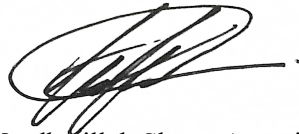
Praise be to Allah SWT for His grace and blessings to carry out one of the obligations as students of the Informatics Department, namely Practical Work (KP).

The author understands that this report still has shortcomings, therefore constructive criticism and suggestions are appreciated. Hopefully, this report can be useful for readers and become a valuable reference in the field of information security. Also, the author hopes that this report can increase the reader's insight and knowledge into the topic of Steganography.

Through this report book, the author would also like to express her gratitude to the people who have helped throughout the internship period and provide guidance to compile the internship report, both directly and indirectly, including:

1. Hudan Studiawan, S.Kom., M.Kom., Ph.D. as the supervisor and coordinator of the internship.
2. Prof. Tohari Ahmad, S.Kom., MIT., Ph.D. as field supervisor during the internship.
3. All senior colleagues in the Network-Based Computing Lab for their assistance and discussions during the implementation of the KP.

Surabaya, 19 Maret 2025



Mardhatillah Shevy Ananti

[Halaman ini sengaja dikosongkan]

CHAPTER 1

INTRODUCTION

1.1. Background

In today's digital era, information security is one of the important aspects to protect data from the threat of theft or wiretapping. Steganography is one of the security techniques used to hide confidential information in digital media, such as audio [1], video [2], and images [3]. By aiming to maintain the confidentiality of data without attracting the attention of unauthorized parties, this technique is widely used in various fields, including confidential communications, copyright protection, and digital forensics. By hiding secret data in ordinary files to keep it invisible to unauthorized people, the steganography technique ensures that it remains hidden and provide security [4]. Image steganography requires a cover image, which is the starting image that is used as the host to store the secret data [5]. Additionally, a stego image is involved as the image containing the secret data that has been altered [6].

Although image steganography provides a solution in data protection, there is a major challenge in its implementation, specifically in balancing the capacity of data insertion (payload) and the quality of the stego image [7], [8]. The quality of the stego image tends to decrease when the amount of data inserted is increased. This can be detected through steganalysis [9], [10]. Therefore, an adaptive steganography method is needed to optimize data insertion without sacrificing image quality.

1.2. Objective

The aim of this internship is to help in the implementation of an image steganography method called FuzzyStego, utilizing fuzzy logic to optimize the data embedding area in digital images. During this internship, I contributed in the implementation of the

algorithm, performance testing, and analysis of the results using evaluation parameters such as Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM). The outcome of this method is to increase the data embedding capacity without lowering the image quality, as well as contributing to the development of more adaptive and secure steganography techniques.

1.3. Benefit

The benefits of this research are both academic and practical. Academically, this research contributes to the development of a more adaptive steganography method by utilizing fuzzy logic to improve the balance between the amount of data inserted and the quality of the image. Practically, the results of this study can be applied in information security, particularly in protecting digital data to make it more difficult to detect by unauthorized parties.

1.4. Formulation of the problem

The problem formulation of this practical work is as follows:

1. How can the quality of the stego images be developed using fuzzy logic?
2. How does FuzzyStego affect PSNR and SSIM compared to traditional steganography methods?
3. How effective is the FuzzyStego method, using fuzzy logic, in adjusting the number of hidden bits?

1.5. Location and Time of Internship

This internship was conducted online as, during that time, the author was undergoing a double degree program at The University of Queensland, Australia. All internship activities, including discussions and guidance, were conducted via Zoom Meeting with

the supervisor and research team at the Net Centric Computing Lab. This internship took place from July 2024 to December 2024.

1.6. Practical Work Methodology

The methodology for creating the research paper includes:

1.6.1. Literature Study

This stage examines previous research on steganography, fuzzy logic and optimization techniques for data insertion in digital images. By analyzing fundamental principles, strengths and limitations, this literature study establishes an understanding of current methods developed in the field. The research evaluates three steganography embedding methods: Transform Domain Methods, Least Significant Bit (LSB) insertion and Pixel Value Differencing (PVD) to assess their payload capacity and image quality performance. Research on fuzzy logic applications in information security demonstrates its capability to improve adaptability and minimize data hiding scheme distortion. Through this literature study, weaknesses including the balance between embedding capacity and imperceptibility, vulnerability to steganalysis attacks and traditional methods to adapt across different types of images, are discussed. FuzzyStego uses fuzzy logic-based adaptive embedding mechanism to enhance both stego image quality and security, which seeks to find weaknesses and to serve in its development.

1.6.2. Algorithm Implementation

After the literature study, the following stage involves the development of FuzzyStego algorithm which uses fuzzy logic to enhance adaptive data embedding in digital images. The algorithm uses a fuzzy system to sort the pixels in the image into five intensity levels which include L, ML, M, MH and H. The classification process determines the number of secret bits that can be embedded into each pixel without causing major changes in the quality of the

image. Pixels with lower intensity values can accommodate more embedded bits without noticeable changes in the image quality, while pixels with higher intensity values are assigned fewer bits to preserve the visual imperceptibility of changes. Both Python and Matlab are utilized for the implementation of the algorithm. The embedding and extraction processes uses Python programming, whereas Matlab was used for data preprocessing and image quality evaluation using PSNR and SSIM metrics.

1.6.3. Performance Testing

At this stage, testing of the implemented algorithm are carried out through images obtained from the SIPI Image Database, a standard research benchmark for steganography and image processing studies. The testing images consisted of grayscale 512×512 pixel images, offering an appropriate combination of computational efficiency and image complexity. To evaluate the performance, experiments were carried out where secret data are embedded into these images using adaptive fuzzy logic-based embedding strategies and then assessed by calculating the PSNR and the SSIM. The PSNR measurement evaluated the embedding distortion by comparing the stego images with the original cover images and the SSIM evaluated structural and perceptual image quality.

1.6.4. Results Analysis

From the previous stage, experimental results were obtained which were evaluated against traditional steganography approaches including LSB insertion and PVD. LSB and PVD are two widely used baseline techniques in image steganography, LSB provides high capacity but suffers from visual distortion while PVD adapts to pixel differences for data embedding. The proposed FuzzyStego method was evaluated through comparative analysis to determine its effectiveness in increasing data insertion capacity while preserving high visual and structural quality of stego images. The evaluation used PSNR and SSIM as performance metrics. This

result analysis shows that FuzzyStego achieves higher embedding capacity and imperceptibility through the integration of fuzzy logic compared to traditional methods.

1.7. Report Systematics

1.7.1. Chapter I Introduction

This chapter contains the background, objectives, benefits, problem formulation, location and time of the practical work, methodology, and systematics of the report.

1.7.2. Chapter II Company Profile

This chapter describes the Net Centric Computing Lab, where the practical work is carried out, including the vision and mission, as well as the research areas carried out in the laboratory.

1.7.3. Chapter III Literature Review

This chapter contains the theoretical basis that supports the research, including the concept of steganography.

1.7.4. Chapter IV System Implementation

This chapter contains a description of the stages carried out for the implementation process of the FuzzyStego algorithm.

1.7.5. Chapter V Testing and Analysis of Results

This chapter contains the results of testing the FuzzyStego method using image datasets, as well as comparative analysis that has been developed during the implementation of the practical work.

1.7.6. Chapter VI Conclusions and Suggestions

This chapter contains conclusions and suggestions obtained from the process of implementing the practical work.

[Halaman ini sengaja dikosongkan]

CHAPTER 2

LABORATORY PROFILE

2.1. Net Centric Computing Laboratory Profile

The Net Centric Computing Lab is one of the research laboratories in the Department of Informatics, Institut Teknologi Sepuluh Nopember (ITS) which focuses on the fields of information security, computer networks, distributed computing, and digital image processing. This laboratory is a research center for various technologies, including steganography, cryptography, and artificial intelligence-based system optimization.

2.2. Location

Department of Informatics Engineering, ITS, Keputih, Sukolilo, Surabaya, East Java 60117

[Halaman ini sengaja dikosongkan]

CHAPTER 3

LITERATURE REVIEW

3.1. Steganography

Steganography is a technique for hiding information in a digital medium, such as images, audio, or video, with the aim of maintaining the confidentiality of the data without attracting the attention of unauthorized parties. Unlike cryptography which encrypts messages so that they cannot be read without a certain key, steganography attempts to hide the existence of the message itself. In the field of digital image processing, steganography is often applied using methods such as LSB, PVD, and Transform Domain Techniques. However, the main challenge in steganography is maintaining a balance between data insertion capacity (payload) and image quality (stego image), because the more data is inserted, the greater the possibility of visual distortion.

3.2. Fuzzy Logic in Steganography

Fuzzy logic is a computational approach that mimics the way humans make decisions based on uncertain or ambiguous information. In the context of steganography, classification of image pixels are utilized with the use of fuzzy logic based on their intensity levels, allowing adaptation of the number of bits inserted according to the visual characteristics of the image. In the FuzzyStego method, pixels are grouped into L, ML, M, MH, and H based on their intensity values. Pixels with low intensity have a higher tolerance for changes, so they can insert more bits without causing visible distortion. In contrast, pixels with high intensity are given only a small amount of data to maintain image quality.

3.3. Relevant Steganography Methods

Several steganography methods that have been developed previously and are the basis for this research include:

3.3.1. Least Significant Bit (LSB)

LSB stands as one of the fundamental and commonly used methods in image steganography. The method operates through substituting the least significant bit of pixel binary values with the bits from secret data. The replacement of least significant bits results in such small pixel intensity changes that human observers cannot detect them thus making LSB an effective yet basic steganographic method. The LSB method provides both high embedding capacity and simple operation yet it suffers from major security weaknesses. The hidden data in LSB becomes highly susceptible to destruction through common image processing operations including compression and noise addition. LSB remains vulnerable to statistical detection through chi-square tests and other analysis techniques which reduces its security value for sensitive applications. The identified limitations in LSB steganography create a need for developing more adaptive and robust steganographic methods including FuzzyStego.

3.3.2. Pixel Value Differentiation (PVD)

PVD is a steganographic technique that operates by splitting images into small pixel blocks to determine data embedding capacity based on pixel value differences between blocks. The image smoothness determines data insertion capacity because small pixel differences result in limited embedding capacity to prevent noticeable distortion. The embedding capacity for data increases when pixel differences become larger because it allows for additional data concealment without detection. The adaptive nature of PVD exceeds traditional methods like LSB insertion because it modifies its approach based on image content to achieve better data hiding efficiency. The quality of images remains a challenge for PVD because excessive data embedding in high-

difference areas produces visible distortions and compression artifacts thus requiring careful balancing of hidden data with image perceptibility.

3.3.3. Transform Domain Techniques

Transform domain techniques embed data into image frequency domains through Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT). These methods operate on image frequency coefficients to modify spatial frequencies instead of working directly with pixel values. The technique provides superior protection than LSB and PVD spatial domain methods because it embeds data in frequency coefficients. The frequency domain data remains resistant to lossy compression algorithms such as JPEG because these algorithms primarily affect pixel-based steganographic techniques. The frequency domain modifications make the technique less detectable through steganalysis methods and less noticeable to human observers. The main disadvantage of transform domain techniques is their increased computational complexity when compared to basic spatial domain methods. The image transformation process into frequency space for data embedding followed by the reverse process to generate the stego-image requires additional processing power and time which reduces efficiency in real-time and resource-constrained applications. Transform domain methods remain popular because they provide enhanced security features and strong resistance against typical image processing methods..

3.4. Steganography Quality Evaluation Parameters

To measure the quality of the steganography image, two main parameters are used, namely the PSNR and the SSIM.

3.4.1. Peak Signal-to-Noise Ratio (PSNR)

PSNR is one of the most widely used metrics to measure the quality difference between the cover image (original image) and

the stego image (image after data embedding). It measures the amount of distortion or noise that is introduced during the embedding process and higher values indicate less distortion and better image quality. PSNR is stated in decibels (dB) and is calculated using the formula below:

$$PSNR = 10 \times \log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

In this formula, MAX_I^2 is the maximum possible pixel value of the image (usually 255 for an 8-bit image) and MSE (Mean Squared Error) is the average of the squared differences between corresponding pixels in the cover image and the stego image. The MSE is a measure of the total error or deviation that is caused by the data embedding process. Since the PSNR value is inversely proportional to MSE, a lower MSE will result in a higher PSNR value which means that the stego image is closer to the original cover image in terms of quality. A higher PSNR is generally preferred since it means that the changes made by the embedding process are less noticeable to the human eye. However, PSNR is a useful indicator of image quality, but it does not always correspond to the visual perception, since it does not take into account the perceptual factors such as contrast or texture. Thus, PSNR should be considered together with other quality metrics in order to evaluate the effectiveness of steganography techniques.

3.4.2. Structural Similarity Index Measure (SSIM)

The SSIM is a perceptual metric that is used to measure the structural similarity between the cover image and the stego image. Unlike PSNR, which only considers pixel-wise differences, SSIM evaluates the image quality by considering three key components: luminance, contrast, and structure, which are crucial for human visual perception. These components allow SSIM to provide a more accurate representation of perceived image quality, as it models the way the human eye perceives changes in images. The SSIM value ranges from 0 to 1, where a value of 1 indicates that

the two images are identical in terms of structure and quality, while a value closer to 0 indicates significant differences between the images. SSIM is calculated using the following formula:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

In this formula, μ represents the mean pixel intensity of the image, σ represents the variance, and σ_{xy} denotes the covariance between the two images. C_1 and C_2 are small constants used to avoid division by zero and ensure numerical stability. The SSIM index evaluates the structural similarity by comparing local patterns of pixel intensities in the image, which are correlated with perceptual image quality. Compared to PSNR, SSIM is generally regarded as more reliable because it better reflects human visual perception, taking into account not just pixel differences but also structural and textural features that are important for the human eye. Therefore, SSIM is particularly useful in image processing tasks like steganography, where maintaining visual quality is important, and provides a more holistic evaluation of the stego image's perceptual fidelity.

[Halaman ini sengaja dikosongkan]

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1. Data Preparation

This chapter explains the stages in implementing the FuzzyStego algorithm. The initial stage is data preparation, which includes dataset selection, preprocessing, and image parameter adjustment for steganography testing.

In this study, an image dataset from the SIPI Image Database was used, which is one of the standard datasets in steganography and image processing research. This dataset consists of various types of grayscale images measuring 512×512 pixels, which were chosen because they provide a balance between visual quality and complexity of image structure in testing steganography algorithms.

4.2. Implementation of FuzzyStego Algorithm

The implementation phase includes the development of the FuzzyStego algorithm, which aims to optimize the data embedding area in digital images using fuzzy logic. This process consists of several main steps, all of which had been compiled into a Methodology Flowchart, in Figure 1. The steps includes the following:

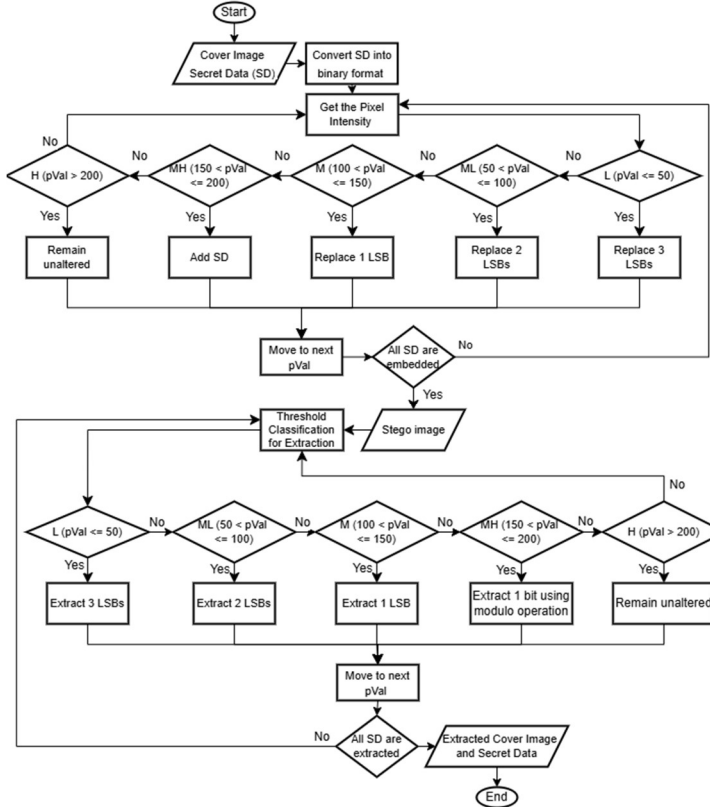


Figure 1. Methodology Flowchart of FuzzyStego Method

4.2.1. Embedding Algorithm

The embedding process starts by importing both the cover image and the secret data. The secret data requires conversion into its binary form before the process begins. The algorithm retrieves pixel intensity values from the cover image before it performs conditional operations to determine the number of secret data bits that can be embedded into each pixel. The algorithm embeds 3 bits of secret data into the 3 least significant bits of the pixel when

intensity values are below or equal to a specified low threshold while performing the necessary pixel update. The embedding process inserts 2 bits of secret data into the 2 least significant bits of pixels when intensity values exist between the low and medium-low thresholds. The embedding process inserts one bit of secret data into the least significant bit when pixel intensities fall between the medium-low and medium thresholds. The process adds a pixel value bit when intensity levels exist between the medium and medium-high thresholds. The pixel remains unaltered when its intensity exceeds the medium-high threshold because no data embedding occurs. The process repeats until all secret data bits find their way into the image. The stego image emerges from this process as the final output because it contains the concealed information.

4.2.2. Extracting Algorithm

The extraction process begins by loading the stego image that contains the embedded secret data. The algorithm examines each pixel in the image for its intensity value and determines how many bits to extract based on that value. If the intensity is less than or equal to a specified low threshold, 3 bits are extracted from the 3 least significant bits of the pixel and added to a list of secret bits. Similarly, if the intensity is between the low and medium-low thresholds, 2 bits are extracted; and if it lies between the medium-low and medium thresholds, only 1 bit is extracted from the least significant bit. In each of these cases, the cover image is also updated by resetting the respective least significant bits that were used for embedding. If the intensity is between the medium and medium-high thresholds, a bit is extracted using a modulo operation and the pixel value is updated accordingly. For pixels with intensities above the medium-high threshold, no data is extracted, and the pixel remains unchanged. This process continues until all the secret data bits have been retrieved. The result of this procedure is the recovered cover image and the extracted secret data.

4.3. Performance Testing and Evaluation

After the implementation of the FuzzyStego algorithm, testing was carried out to evaluate the effectiveness of this method compared to conventional steganography techniques.

4.3.1. Testing Methods

Once the embedding process is complete, a stego image is generated and compared to the cover image to ensure minimal visual changes:

- Dataset: 512×512 grayscale images from SIPI Image Database.
- Evaluation parameters:
 - PSNR: Measures the difference between the cover image and the stego image.
 - SSIM: Measures the degree of similarity of image structure after data embedding.

4.3.2. Pseudocodes of Matlab Functions

By using Matlab, configuration of functions are implemented to test out FuzzyStego.

The first function, *Function 1*, involves the embedding algorithm by modifying the number of embedded bits per pixel according to its intensity level. The algorithm embeds more bits into pixels with low intensity but fewer bits into pixels with high intensity to achieve maximum capacity while maintaining image quality. See Attachment, Figure 7, for the whole code.

Function 1: embedAndCalculatePSNR

Input: Cover Image, Secret Data Path

Load the cover image from the specified path.

Read binary digits (0 or 1) from the secret data file.

Convert the binary digits to ASCII values, then to 8-bit binary per character, and flatten into a 1D bit array.

Set intensity thresholds: $tL = 50$, $tML = 100$, $tM = 150$, $tMH = 200$.

Copy the cover image to `stegoImage` and initialize `embeddedBitsCount` to 0.

Start the timer to measure embedding time.

For each pixel in the cover image:

Get the pixel intensity.

If $\text{intensity} \leq tL$ and 3 bits are available, replace the 3 LSBs with the next 3 secret bits and update counters.

Else if $\text{intensity} \leq tML$ and 2 bits are available, replace the 2 LSBs with the next 2 secret bits and update counters.

Else if $\text{intensity} \leq tM$ and 1 bit is available, replace the LSB with the next secret bit and update counters.

Else if the pixel intensity is greater than the medium-low threshold and less than or equal to the medium threshold then

Else if $\text{intensity} \leq tMH$ and 1 bit is available, add the bit value to the pixel and update counters.

Else if $\text{intensity} > tMH$ and 1 bit is available, add the bit value to the pixel and update counters.

End the loop after processing all pixels.

Stop the timer and store the elapsed time.

Save the stego image in TIFF format to the specified output path.

Calculate PSNR, MSE, and SSIM between the stego and cover images.

Compute the total number of embedded bits and calculate embedding capacity in bits per pixel.

Display the PSNR, MSE, SSIM, total embedded bits, embedding capacity (bpp), and embedding time.

Output: Stego Image, PSNR, MSE, SSIM, Embedding Capacity, Embedding Time

The second function, *Function 2*, implements the extraction Algorithm to perform the reverse embedding operation by examining pixel intensity to determine the number of bits that need extraction. The extraction process recovers the secret data through the same fuzzy thresholds which were used during embedding. See Attachment, *Figure 8*, for the whole code.

Function 2: embedAndCalculatePSNRAndExtract

Input: Stego Image

Load the stego image from the specified path.

Initialize an empty list to store extracted secret bits.

Set intensity thresholds: $tL = 50$, $tML = 100$, $tM = 150$, $tMH = 200$.

For each pixel in the stego image:

Get the pixel intensity.

If intensity $\leq tL$, extract 3 bits from the 3 LSBs and append to the secret bits list.

Else if intensity $\leq tML$, extract 2 bits from the 2 LSBs and append to the secret bits list.

Else if intensity $\leq tM$, extract 1 bit from the LSB and append to the secret bits list.

Else if intensity $\leq tMH$, extract 1 bit using modulo operation (e.g., mod 2) and append to the secret bits list.

Else if intensity $> tMH$, skip extraction; no bits are embedded in high-intensity pixels.

End the loop once all pixels are processed or the desired length of data is recovered.

Reconstruct the cover image by resetting modified LSBs based on embedding rules if necessary.

Convert the list of extracted bits into characters using 8-bit binary to ASCII conversion.

Return the recovered cover image and the extracted secret data.

Output: Reconstructed Cover Image, Extracted Secret Data

Next, *Function 3*, performs a no-fuzzy embedding algorithm as a basic benchmark method which replaces the 2 LSBs of all pixels with secret data bits at a uniform rate. The direct comparison between fuzzy-based and non-fuzzy embedding methods becomes possible through this method to evaluate their imperceptibility and efficiency. See Attachment, *Figure 9*, for the whole code.

Function 3: embedAndCalculatePSNR_NoFuzzy

Input: Cover Image, Secret Data

Load the cover image from the specified path.

Initialize an empty list to store extracted secret bits.

Open the secret data text file and read binary digits (0 or 1).

Convert the digits to ASCII values, then to 8-bit binary representation.

Flatten the binary matrix into a 1D bit array.

Copy the cover image to a new variable as the stego image.

Initialize embeddedBitsCount = 0 and bitIndex = 1.

For each pixel in the cover image:

If at least 2 bits remain, replace the 2 LSBs of the pixel with the next 2 bits from the secret bit array.

Update bitIndex and embeddedBitsCount accordingly.

End the loop when all pixels are processed or secret bits are exhausted.

Save the stego image in TIFF format.

Compute PSNR, MSE, and SSIM between the cover and stego images.

Calculate embedding capacity in bits and bits per pixel (bpp).

Output: Stego Image, Embedding Metrics

Lastly, *Function 4*, a compression-based embedding algorithm which uses secret data compression before the embedding operations begin. The compression process reduces embedded bit numbers which results in higher capacity and less distortion. See Attachment, *Figure 10*, for the whole code.

Function 4: embedAndCalculatePSNRCompression

Input: Cover Image, Compressed Secret Data

Load the cover image.

Open and read compressed secret data in binary form.

Convert binary to ASCII, then to 8-bit binary form and flatten into a 1D array.

*Copy the cover image to stego image and initialize
embeddedBitsCount = 0, bitIndex = 1.*

*Define intensity thresholds: $tL = 50$, $tML = 100$, $tM = 150$,
 $tMH = 200$.*

Start the embedding timer.

For each pixel in the cover image:

For each pixel in the cover image:

*If intensity $\leq tL$ and at least 3 bits available, replace 3
LSBs with next 3 bits.*

*Else if intensity $\leq tML$ and at least 2 bits available,
replace 2 LSBs with next 2 bits.*

*Else if intensity $\leq tM$ and at least 1 bit available, replace
1 LSB with the next bit.*

Else if intensity $\leq t_{MH}$ and at least 1 bit available, add the bit (as integer) to the pixel value.

Else if intensity $> t_{MH}$ and at least 1 bit available, also add the bit to the pixel value.

Update bitIndex and embeddedBitsCount after each embedding.

End the loop or stop when secret bits are fully embedded.

Stop the timer.

Save the stego image in TIFF format.

Compute PSNR, MSE, and SSIM.

Calculate total embedded bits and bits per pixel.

Output: Stego Image, Embedding Metrics

[Halaman ini sengaja dikosongkan]

CHAPTER 5

TESTING AND ANALYSIS OF RESULTS

5.1. Experimental Results

The experimental results in Table 1 demonstrate that FuzzyStego maintains high visual quality in stego images. The method reached an average PSNR value of 58.63 dB while maintaining SSIM values near 1.00 which indicates that embedded images remain almost unnoticeable to the human eye, when compared to original cover images. The FuzzyStego method maintains high visual quality during the use of higher payload sizes.

Table 1. PSNR and SSIM Results of FuzzyStego

Images	20kb		40kb		60kb		80kb		100kb	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Aerial	59.62	1.00	55.87	1.00	53.57	1.00	51.68	1.00	51.84	1.00
Airplane	55.25	1.00	58.21	1.00	58.21	1.00	58.21	1.00	58.21	1.00
Car and APCs	51.19	1.00	48.14	1.00	46.70	0.99	45.83	0.99	45.27	0.99
Fishing Boat	57.46	1.00	52.22	1.00	48.01	1.00	45.31	0.99	44.42	0.99
Pixel ruler	58.28	1.00	55.85	1.00	53.92	1.00	52.43	1.00	53.56	1.00
Stream and bridge	56.39	1.00	53.10	1.00	49.41	1.00	47.90	1.00	46.73	1.00
Tank	56.19	1.00	52.27	1.00	49.73	1.00	48.13	1.00	47.77	1.00
Truck	52.02	1.00	49.17	1.00	47.45	1.00	46.11	1.00	45.52	1.00
Lena	56.64	1.00	53.34	1.00	51.15	1.00	49.54	1.00	46.16	1.00
Peppers	52.56	1.00	49.20	1.00	47.20	1.00	45.72	0.99	44.42	0.99
Barbara	55.70	1.00	52.14	1.00	50.76	1.00	49.40	1.00	48.19	0.99
Zelda	54.10	1.00	51.06	1.00	49.25	0.99	48.28	0.99	47.67	0.99
Baboon	51.43	1.00	49.22	1.00	47.74	1.00	46.73	1.00	46.16	1.00

The average values of PSNR and SSIM from Table 2 demonstrate FuzzyStego's resistance to compression attacks when applied to both compressed cover and stego images. Huffman compression method represents a widely used algorithm which produces significant distortion. The PSNR values of the compressed stego images, Stream and Bridge, range from 33.73 to 41.14 for the Pixel Ruler image. The stego images demonstrate excellent quality retention following compression. The PSNR value of 41.14 for the Pixel Ruler image indicates low distortion while the 33.74 value for Stream and Bridge indicates slightly higher distortion. The obtained values remain suitable for typical operational needs. The SSIM values demonstrate high structural similarity between original and compressed images across all test images.

Table 2. Huffman Compression PSNR and SSIM Results.

Test Image	Average PSNR	Average SSIM
Aerial	34.53	0.94
Airplane	38.99	0.94
Car and APCs	35.11	0.90
Fishing Boat	35.12	0.91
Pixel ruler	41.14	0.99
Stream and bridge	33.734	0.95
Tank	34.41	0.88
Truck	34.96	0.90

5.2. Comparative Analysis with Previous Methods

To measure the superiority of the FuzzyStego method, the experimental results were compared with existing methods, shown in *Figure 2*. As displayed, a PSNR value of 52.43 dB for the Boat image and 50.72 dB for the Baboon image, demonstrates that the FuzzyStego method has a strong performance. Obtained by FuzzyStego, the PSNR values exceed the existing methods [11, 12], presenting values less than 42 dB for both images. The proposed method shows exceptional quality preservation capabilities after embedding data. FuzzyStego maintains competitive performance against methods from [13, 14, 15]. The PSNR value of FuzzyStego (52.43 dB) exceeds the PSNR value of [13] 46.30 dB, revealing its ability to embed data with minimal visual distortion. The PSNR values of FuzzyStego (50.72 dB) exceed those of [14, 15] 58.70 dB and 59.11 dB but FuzzyStego maintains an excellent trade-off between image quality and embedding efficiency. The methods from [11, 13, 15] produce Baboon image PSNR values below 48 dB which indicates significant image degradation. The PSNR value of FuzzyStego reaches 50.72 dB which results in better visual quality preservation. The PSNR values of methods [14, 15] reach 54.27 dB and 55.45 dB respectively.

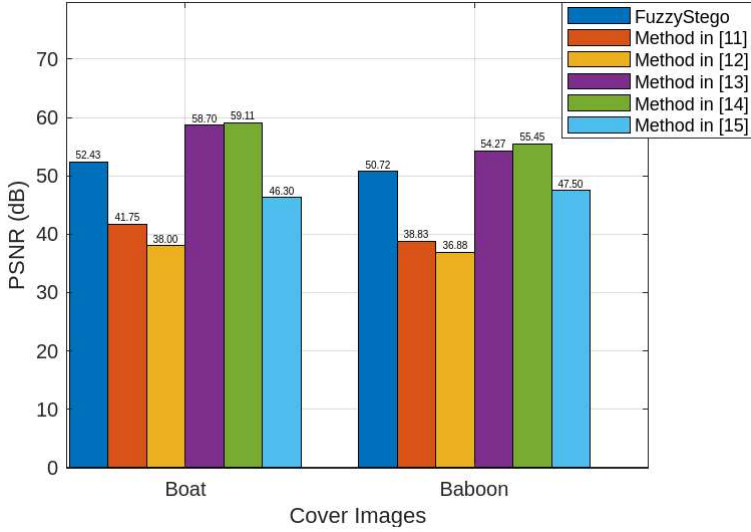


Figure 2. PSNR Results of FuzzyStego and Existing Algorithms

Figure 3 shows the significant superiority of FuzzyStego method compared to other existing algorithms, displaying the maximum embedding capacity achieved on Boat and Baboon cover images. FuzzyStego consistently gives the best results, recording the highest embedding capacity in all tests. This performance indicates that FuzzyStego is very suitable for steganography applications that require high capacity. For the Boat image, FuzzyStego successfully embeds up to 1,079,052 bits, far surpassing other methods. Although the method in [11] is in second place with 824,789 bits, its value is still about 23.6% lower. Meanwhile, the methods from [14, 15] are only able to embed 34,059 bytes and 499,992 bytes, which means a decrease of more than 96.8% and 53.7% compared to FuzzyStego. These findings indicate a significant performance difference between FuzzyStego and other methods. Similar results are also seen in the Baboon image, where FuzzyStego achieves a capacity of 1,078,190 bits. Method [11] is next with 793,183 bits, about 26.5% lower.

Methods [14, 15] show much lower results, namely 17,582 bits and 499,995 bits respectively, indicating a reduction of 98.4% and 53.7%.

Figure 4 shows the histograms of two test images, namely one general image and one medical image, both before and after the embedding process using the FuzzyStego method. The comparative analysis of the histograms shows a very high similarity between the cover image and the stego image, indicating that FuzzyStego is able to embed data efficiently without disturbing the visual appearance of the original image. The very small difference in distribution between the two histograms confirms the ability of this method to hide information well, so that the stego image looks almost identical to the cover image. This shows the reliability of FuzzyStego and supports its application in real-world situations.

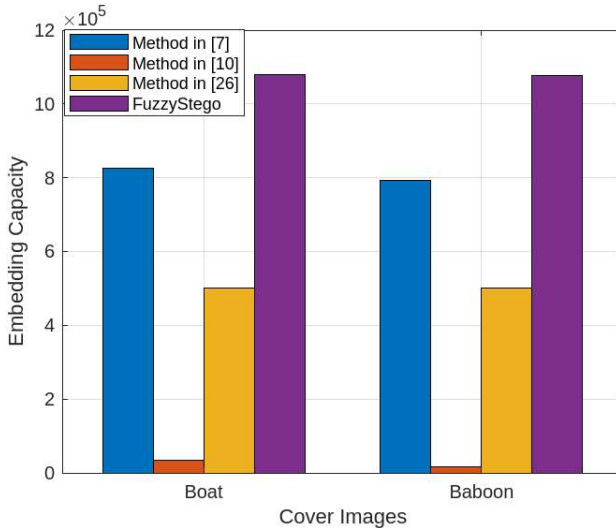


Figure 3. EC Results of FuzzyStego and Existing Algorithms

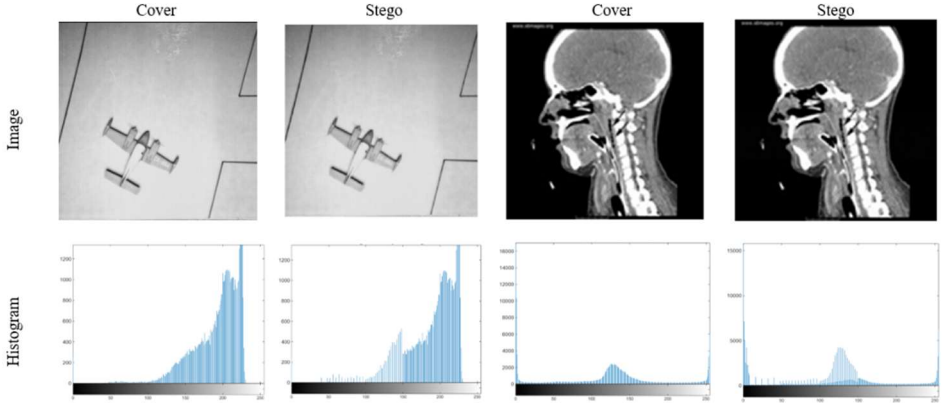


Figure 4. Cover and Stego Images using FuzzyStego Histogram

5.3. Evaluation of the Effectiveness of the FuzzyStego Method

Based on the experimental results, the FuzzyStego method is proven to be superior in protecting the stego images' quality compared to traditional methods. Illustrates in *Figure 5*, the PSNR results demonstrate the comparative performance between the fully proposed FuzzyStego and the NoFuzzy, which is the method without fuzzy logic for adaptive embeddable pixel selection. The PSNR results shows that FuzzyStego outperforms the LSB method according to the figure. The PSNR values of FuzzyStego remain high at all test image levels when the payload size is 20 KB or less and reach above 65 dB. The images of Aerial and Airplane achieve PSNR values of around 70 dB while the LSB method produces values below 60 dB which indicates significant degradation. The PSNR values of FuzzyStego decrease gradually when the payload reaches medium levels between 20–60 KB but stay above 50 dB which shows its scalability. The NoFuzzy method shows a rapid decline in PSNR values which results in PSNR measurements below 40 dB for Brain and Hand images. FuzzyStego maintains high PSNR values above 40 dB at all payload sizes above 80 KB while the NoFuzzy approach produces low quality stego images with PSNR values below 30 dB. The system shows resistance to

large data payloads because it maintains image quality at high levels which meets essential requirements for real-world applications.

Whereas, in

Figure 6 includes the SSIM as a function of payload size under two general purposes and medical images. FuzzyStego maintains SSIM values above 0.98 because it preserves structural fidelity better than the NoFuzzy approach. The SSIM values of FuzzyStego decrease slightly with increasing payload but stay within practical usage thresholds. The NoFuzzy approach shows a major decline in SSIM measurements when the payload size increases. The SSIM value of the Brain image falls below 0.97 when the payload reaches 50 KB and higher which indicates substantial structural degradation. The SSIM curves of FuzzyStego images including Aerial and Airplane remain flat even when the payload reaches its maximum size. The SSIM curves of NoFuzzy demonstrate steep declines which demonstrate its poor ability to maintain image structure. The basic embedding approach of NoFuzzy causes substantial image quality deterioration and structural damage when payload sizes grow larger. FuzzyStego uses its intelligent embedding system to provide scalable performance which preserves both image quality and structural integrity at all payload levels.

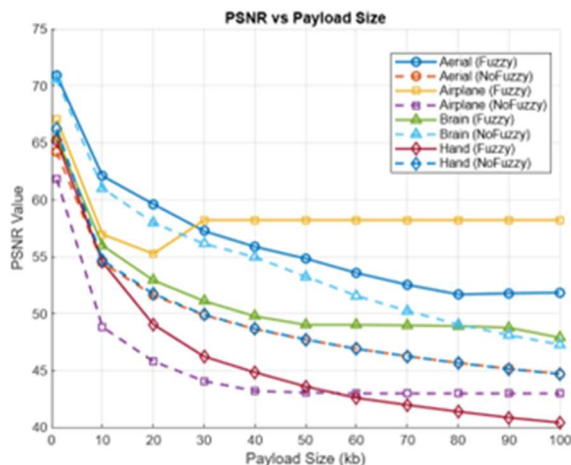


Figure 5. Ablation Experiments Results through PSNR Results

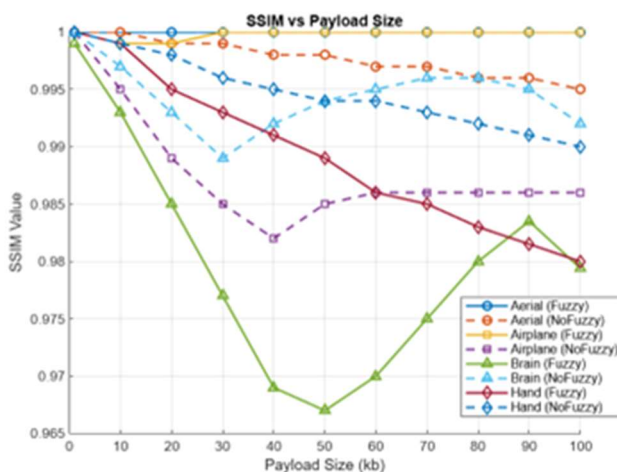


Figure 6. Ablation Experiments Results through SSIM Results

[Halaman ini sengaja dikosongkan]

CHAPTER 6

CONCLUSION AND SUGGESTIONS

7.1. Conclusion

As demonstrated from the results of the FuzzyStego research, it can be concluded that fuzzy logic contributes significantly to improve image quality and maximize data embedding capacity in stego images. The adaptive fuzzy inference system assesses pixel intensity to categorize them into Low, Medium-Low, Medium, Medium-High and High levels and determines the secret bit embedding amount for each pixel based on these classifications. The adaptive approach of the method minimizes noticeable visual distortion which creates stego images that are indistinguishable from the original cover images to both human observers and objective quality assessment methods.

The FuzzyStego method performs better than LSB and PVD traditional steganographic techniques due to the fuzzy logic-based pixel selection and bit allocation strategy. This method directly attributes to the embedding process, optimizing it based on the image's local characteristics instead of applying random insertion. The method achieves exceptional results as it maintains both visual and structural image quality with a PSNR of 58.64 dB and SSIM approaching 1.00. The PSNR values and visual artifacts of LSB and PVD become lower at high payload capacities yet FuzzyStego produces better results.

Furthermore, the FuzzyStego method achieves its effectiveness through its adaptive mechanism which adjusts the amount of secret bits inserted according to pixel intensity levels. The FuzzyStego method enables adaptive bit concealment through pixel intensity-based bit allocation. The method uses pixels with low intensity values for maximum three-bit embedding but restricts pixels with higher intensity values to single bits or minimal changes to maintain visual quality. The fuzzy rule sets and functions control a dynamic bit allocation process which allows

the method to achieve greater embedding capacity without compromising image quality. Compared to traditional methods, FuzzyStego offers a smart and adaptive solution that addresses the challenges of image steganography.

7.2. Suggestion

Although the FuzzyStego method has shown promising results, there are still several aspects that can be further developed, including:

1. Testing with a wider dataset: Using a more diverse dataset, including color and higher resolution images, can provide a deeper understanding of the performance of this method.
2. Improved resistance to steganalysis: Further development is needed to test how resistant this method is to more complex steganalysis attacks.
3. Algorithm optimization for computational efficiency: Currently, the use of fuzzy logic in steganography still requires additional computational processes. Increasing algorithm efficiency can speed up the embedding and extraction process without reducing the quality of the stego image.
4. Application to other domains: In addition to digital images, this method can be tested on other media such as audio or video to explore the potential for broader applications in data security.

[Halaman ini sengaja dikosongkan]

BIBLIOGRAPHY

- [1] Zhang, X., Li, C., & Tian, L. (2023). Advanced audio coding steganography algorithm with distortion minimization model based on audio beat. *Computers and Electrical Engineering*, 106. <https://doi.org/10.1016/j.compeleceng.2023.108580>
- [2] Debnath, S., Mohapatra, R. K., & Dash, R. (2023). Secret data sharing through coverless video steganography based on bit plane segmentation. *Journal of Information Security and Applications*, 78. <https://doi.org/10.1016/j.jisa.2023.103612>
- [3] Song, B., Wei, P., Wu, S., Lin, Y., & Zhou, W. (2024). A survey on Deep-Learning-based image steganography. In *Expert Systems with Applications* (Vol. 254). Elsevier Ltd. <https://doi.org/10.1016/j.eswa.2024.124390>
- [4] Hu, K., Wang, M., Ma, X., Chen, J., Wang, X., & Wang, X. (2024). Learning-based image steganography and watermarking: A survey. In *Expert Systems with Applications* (Vol. 249). Elsevier Ltd. <https://doi.org/10.1016/j.eswa.2024.123715>
- [5] Rahman, S., Uddin, J., Zakarya, M., Hussain, H., Khan, A. A., Ahmed, A., & Haleem, M. (2023). A Comprehensive Study of Digital Image Steganographic Techniques. *IEEE Access*, 11, 6770–6791. <https://doi.org/10.1109/ACCESS.2023.3237393>
- [6] Khan, M., & Rasheed, A. (2023). A high-capacity and robust steganography algorithm for quantum images. *Chinese Journal of Physics*, 85, 89–103. <https://doi.org/10.1016/j.cjph.2023.06.016>
- [7] Li, Q., Yan, B., Li, H., & Chen, N. (2018). Separable reversible data hiding in encrypted images with improved security and capacity. *Multimedia Tools and Applications*, 77(23), 30749–30768. <https://doi.org/10.1007/s11042-018-6187-y>
- [8] de La Croix, N. J., & Ahmad, T. (2023). Toward secret data location via fuzzy logic and convolutional neural network.

- Egyptian Informatics Journal, 24(3).
<https://doi.org/10.1016/j.eij.2023.05.010>
- [9] Reinel, T. S., Brayan, A. A. H., Alejandro, B. O. M., Alejandro, M. R., Daniel, A. G., Alejandro, A. G. J., Buenaventura, B. J. A., Simon, O. A., Gustavo, I., & Raul, R. P. (2021). GBRAS-Net: A Convolutional Neural Network Architecture for Spatial Image Steganalysis. *IEEE Access*, 9, 14340–14350.
<https://doi.org/10.1109/ACCESS.2021.3052494>
- [10] Ntivuguruzwa, J. D. L. C., & Ahmad, T. (2023). A convolutional neural network to detect possible hidden data in spatial domain images. *Cybersecurity*, 6(1).
<https://doi.org/10.1186/s42400-023-00156-x>
- [11] Sahu, A. K., & Swain, G. (2019). An Optimal Information Hiding Approach Based on Pixel Value Differencing and Modulus Function. *Wireless Personal Communications*, 108(1), 159–174. <https://doi.org/10.1007/s11277-019-06393-z>
- [12] Fahim, A., & Raslan, Y. (2023). Optimized steganography techniques based on PVDS and genetic algorithm. *Alexandria Engineering Journal*, 85, 245–260.
<https://doi.org/10.1016/j.aej.2023.11.013>
- [13] Wu, H., Li, X., Zhao, Y., & Ni, R. (2020). Improved PPVO-based high-fidelity reversible data hiding. *Signal Processing*, 167.
<https://doi.org/10.1016/j.sigpro.2019.107264>
- [14] Chang, J., Ding, F., Li, X., & Zhu, G. (2021). Hybrid prediction-based pixel-value-ordering method for reversible data hiding. *Journal of Visual Communication and Image Representation*, 77.
<https://doi.org/10.1016/j.jvcir.2021.103097>
- [15] Ramadhan, I. F., Anandha, D. A., D'Layla, A. W. C., de La Croix, N. J., & Ahmad, T. (2024). Image Steganography using Customized Differences between the Neighboring Pixels. *Proceedings - International Conference on Informatics*

and Computational Sciences, 496–501.
<https://doi.org/10.1109/ICICoS62600.2024.10636936>

[Halaman ini sengaja dikosongkan]

ATTACHMENT

```
function embedAndCalculatePSNR(coverImagePath, secretDataPath,
stegoImagePath)
    % Load the cover image
    coverImage = imread(coverImagePath);

    % Read the secret data from a text file
    fileID = fopen(secretDataPath, 'r');
    secretDataBits = fscanf(fileID, '%ld');
    fclose(fileID);

    % Convert secret text to its ASCII values and then to binary
    secretData = double(secretDataBits); % Convert to ASCII numeric values
    secretBits = dec2bin(secretData, 8) - '0'; % Convert to binary, 8 bits per
character
    secretBits = secretBits(:)'; % Flatten to a row vector

    % Define thresholds for classification
    tL = 50; % Threshold for Low intensity
    tML = 100; % Threshold for Medium-Low intensity
    tM = 150; % Threshold for Medium intensity
    tMH = 200; % Threshold for Medium-High intensity

    % Initialize the stego image
    stegoImage = coverImage;

    % Initialize the embedded bits counter
    embeddedBitsCount = 0;

    % Start timer for embedding process
    tic;

    % Embed the secret data
    bitIndex = 1;
    for i = 1:size(coverImage, 1)
        for j = 1:size(coverImage, 2)
            pixelValue = coverImage(i, j);

            if pixelValue <= tL
                % L: Replace 3 LSBs
                if bitIndex + 2 <= length(secretBits)
                    stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
```

```

1));          stegoImage(i, j) = bitset(stegoImage(i, j), 2, secretBits(bitIndex +
2));          stegoImage(i, j) = bitset(stegoImage(i, j), 3, secretBits(bitIndex +
3 bits        bitIndex = bitIndex + 3;
               embeddedBitsCount = embeddedBitsCount + 3; % Increment by
               end

               elseif pixelValue > tL && pixelValue <= tML
               % ML: Replace 2 LSBs
               if bitIndex + 1 <= length(secretBits)
               stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
               stegoImage(i, j) = bitset(stegoImage(i, j), 2, secretBits(bitIndex +
1));          bitIndex = bitIndex + 2;
               embeddedBitsCount = embeddedBitsCount + 2; % Increment by
2 bits        end

               elseif pixelValue > tML && pixelValue <= tM
               % M: Replace 1 LSB
               if bitIndex <= length(secretBits)
               stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
               bitIndex = bitIndex + 1;
               embeddedBitsCount = embeddedBitsCount + 1; % Increment by
1 bit        end

               elseif pixelValue > tM && pixelValue <= tMH
               % MH: Add secret data
               if bitIndex <= length(secretBits)
               stegoImage(i, j) = pixelValue +
bin2dec(num2str(secretBits(bitIndex)));
               bitIndex = bitIndex + 1;
               embeddedBitsCount = embeddedBitsCount + 1; % Increment by
1 bit        end

               else
               % H: Add secret data
               if bitIndex <= length(secretBits)

```

```

        stegoImage(i, j) = pixelValue +
        bin2dec(num2str(secretBits(bitIndex)));
        bitIndex = bitIndex + 1;
        embeddedBitsCount = embeddedBitsCount + 1; % Increment by
1 bit
    end
end
end
end

% Stop timer for embedding process
embeddingTime = toc;

% Save the stego image
saveStegoImage(stegoImage, stegoImagePath);

% Calculate PSNR, MSE, and SSIM
psnrValue = psnr(stegoImage, coverImage);
mseValue = immse(stegoImage, coverImage);
ssimValue = ssim(stegoImage, coverImage);

% Calculate embedding capacity (in bits and bits per pixel)
totalPixels = numel(coverImage);
embeddingCapacityBits = embeddedBitsCount;
embeddingCapacityBpp = embeddingCapacityBits / totalPixels;

% Display the results
fprintf('PSNR: %.4f dB\n', psnrValue);
fprintf('MSE: %.4f\n', mseValue);
fprintf('SSIM: %.4f\n', ssimValue);
fprintf('Total embedded bits: %d bits\n', embeddingCapacityBits);
fprintf('Embedding capacity: %.4f bits per pixel (bpp)\n',
embeddingCapacityBpp);
fprintf('Embedding time: %.4f seconds\n', embeddingTime);
end

function saveStegoImage(stegoImage, outputPath)
% Ensure the output file path has a .tiff extension
[~,~, ext] = fileparts(outputFilePath);
if ~strcmp(ext, '.tiff') && ~strcmp(ext, '.tif')
    outputFilePath = strcat(outputFilePath, '.tiff');
end

```

```

% Save the stego image in TIFF format
imwrite(stegoImage, outputFilePath, 'tiff');
fprintf('Stego image saved as %s\n', outputFilePath);
end

```

Figure 7. Embedding and Calculating PSNR

```

function embedAndCalculatePSNRAndExtract(coverImagePath,
secretDataPath, stegoImagePath)
% Load the cover image
coverImage = imread(coverImagePath);

% Read the secret data from a text file
fileID = fopen(secretDataPath, 'r');
secretDataBits = fscanf(fileID, '%ld');
fclose(fileID);

% Convert secret text to its ASCII values and then to binary
secretData = double(secretDataBits); % Convert to ASCII numeric values
secretBits = dec2bin(secretData, 8) - '0'; % Convert to binary, 8 bits per
character
secretBits = secretBits(:)'; % Flatten to a row vector

% Define thresholds for classification
tL = 50; % Threshold for Low intensity
tML = 100; % Threshold for Medium-Low intensity
tM = 150; % Threshold for Medium intensity
tMH = 200; % Threshold for Medium-High intensity

% Initialize the stego image
stegoImage = coverImage;

% Initialize the embedded bits counter
embeddedBitsCount = 0;

% Start timer for embedding process
tic;

% Embed the secret data
bitIndex = 1;
for i = 1:size(coverImage, 1)
    for j = 1:size(coverImage, 2)

```

```

pixelValue = coverImage(i, j);

if pixelValue <= tL
    % L: Replace 3 LSBs
    if bitIndex + 2 <= length(secretBits)
        stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
        stegoImage(i, j) = bitset(stegoImage(i, j), 2, secretBits(bitIndex +
1));
        stegoImage(i, j) = bitset(stegoImage(i, j), 3, secretBits(bitIndex +
2));
        bitIndex = bitIndex + 3;
        embeddedBitsCount = embeddedBitsCount + 3;
    end

elseif pixelValue > tL && pixelValue <= tML
    % ML: Replace 2 LSBs
    if bitIndex + 1 <= length(secretBits)
        stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
        stegoImage(i, j) = bitset(stegoImage(i, j), 2, secretBits(bitIndex +
1));
        bitIndex = bitIndex + 2;
        embeddedBitsCount = embeddedBitsCount + 2;
    end

elseif pixelValue > tML && pixelValue <= tM
    % M: Replace 1 LSB
    if bitIndex <= length(secretBits)
        stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
        bitIndex = bitIndex + 1;
        embeddedBitsCount = embeddedBitsCount + 1;
    end

elseif pixelValue > tM && pixelValue <= tMH
    % MH: Add secret data
    if bitIndex <= length(secretBits)
        stegoImage(i, j) = pixelValue +
bin2dec(num2str(secretBits(bitIndex)));
        bitIndex = bitIndex + 1;
        embeddedBitsCount = embeddedBitsCount + 1;
    end

else
    % H: Add secret data

```

```

        if bitIndex <= length(secretBits)
            stegoImage(i, j) = pixelValue +
bin2dec(num2str(secretBits(bitIndex)));
            bitIndex = bitIndex + 1;
            embeddedBitsCount = embeddedBitsCount + 1;
        end
    end
end
end

% Stop timer for embedding process
embeddingTime = toc;

% Save the stego image
saveStegoImage(stegoImage, stegoImagePath);

% Calculate PSNR, MSE, and SSIM
psnrValue = psnr(stegoImage, coverImage);
mseValue = immse(stegoImage, coverImage);
ssimValue = ssim(stegoImage, coverImage);

% Calculate embedding capacity (in bits and bits per pixel)
totalPixels = numel(coverImage);
embeddingCapacityBits = embeddedBitsCount;
embeddingCapacityBpp = embeddingCapacityBits / totalPixels;

% Extract secret data and measure extraction time
[extractedBits, extractionTime] =
extractSecretDataAndMeasureTime(stegoImagePath);

% Display the embedding and extraction results
fprintf('Embedding Results:\n');
fprintf('PSNR: %.4f dB\n', psnrValue);
fprintf('MSE: %.4f\n', mseValue);
fprintf('SSIM: %.4f\n', ssimValue);
fprintf('Total embedded bits: %d bits\n', embeddingCapacityBits);
fprintf('Embedding capacity: %.4f bits per pixel (bpp)\n',
embeddingCapacityBpp);
fprintf('Embedding time: %.4f seconds\n\n', embeddingTime);

fprintf('Extraction Results:\n');
fprintf('Extraction time: %.4f seconds\n', extractionTime);
fprintf('Number of extracted bits: %d\n', length(extractedBits));

```

```

end

function [secretBits, extractionTime] =
extractSecretDataAndMeasureTime(stegoImagePath)
    % Start the timer to measure extraction time
    tic;

    % Load the stego image
    stegoImage = imread(stegoImagePath);

    % Initialize variables
    secretBits = []; % To store extracted secret bits

    % Define thresholds for classification
    tL = 50; % Threshold for Low intensity
    tML = 100; % Threshold for Medium-Low intensity
    tM = 150; % Threshold for Medium intensity
    tMH = 200; % Threshold for Medium-High intensity

    % Loop through each pixel in the stego image
    for i = 1:size(stegoImage, 1)
        for j = 1:size(stegoImage, 2)
            pixelValue = stegoImage(i, j);

            % Classify the pixel intensity and extract secret bits
            if pixelValue <= tL
                % Low intensity: Extract 3 LSBs
                secretBits = [secretBits, bitget(pixelValue, 1), bitget(pixelValue,
2), bitget(pixelValue, 3)];
            elseif pixelValue > tL && pixelValue <= tML
                % Medium-Low intensity: Extract 2 LSBs
                secretBits = [secretBits, bitget(pixelValue, 1), bitget(pixelValue,
2)];
            elseif pixelValue > tML && pixelValue <= tM
                % Medium intensity: Extract 1 LSB
                secretBits = [secretBits, bitget(pixelValue, 1)];
            elseif pixelValue > tM && pixelValue <= tMH
                % Medium-High intensity: Use modulo to extract bit
                secretBits = [secretBits, mod(pixelValue, 2)];
            else
                % High intensity: Use modulo to extract bit
                secretBits = [secretBits, mod(pixelValue, 2)];
            end
        end
    end
end

```



```

        end
    end
end

% Stop the timer and calculate the extraction time
extractionTime = toc;
end

function saveStegoImage(stegoImage, outputPath)
% Ensure the output file path has a .tiff extension
[~,~, ext] = fileparts(outputFilePath);
if ~strcmp(ext, '.tiff') && ~strcmp(ext, '.tif')
    outputFilePath = strcat(outputFilePath, '.tiff');
end

% Save the stego image in TIFF format
imwrite(stegoImage, outputPath, 'tiff');
fprintf('Stego image saved as %s\n', outputPath);
end

```

Figure 8. Extracting and Calculating PSNR

```

function embedAndCalculatePSNRCompression(coverImagePath,
secretDataPath)
% Load the cover image
coverImage = imread(coverImagePath);

% Read the secret data from a text file
fileID = fopen(secretDataPath, 'r');
secretDataBits = fscanf(fileID, '%ld');
fclose(fileID);

% Convert secret text to its ASCII values and then to binary
secretData = double(secretDataBits); % Convert to ASCII numeric values
secretBits = dec2bin(secretData, 8) - '0'; % Convert to binary, 8 bits per
character
secretBits = secretBits(:)'; % Flatten to a row vector

% Define thresholds for classification
tL = 50; % Threshold for Low intensity
tML = 100; % Threshold for Medium-Low intensity
tM = 150; % Threshold for Medium intensity

```

```

tMH = 200; % Threshold for Medium-High intensity

% Initialize the stego image
stegoImage = coverImage;

% Embed the secret data
bitIndex = 1;
for i = 1:size(coverImage, 1)
    for j = 1:size(coverImage, 2)
        pixelValue = coverImage(i, j);

        if pixelValue <= tL
            % L: Replace 3 LSBs
            if bitIndex + 2 <= length(secretBits)
                stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
                stegoImage(i, j) = bitset(stegoImage(i, j), 2, secretBits(bitIndex +
1));
                stegoImage(i, j) = bitset(stegoImage(i, j), 3, secretBits(bitIndex +
2));
                bitIndex = bitIndex + 3;
            end

        elseif pixelValue > tL && pixelValue <= tML
            % ML: Replace 2 LSBs
            if bitIndex + 1 <= length(secretBits)
                stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
                stegoImage(i, j) = bitset(stegoImage(i, j), 2, secretBits(bitIndex +
1));
                bitIndex = bitIndex + 2;
            end

        elseif pixelValue > tML && pixelValue <= tM
            % M: Replace 1 LSB
            if bitIndex <= length(secretBits)
                stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
                bitIndex = bitIndex + 1;
            end

        elseif pixelValue > tM && pixelValue <= tMH
            % MH: Add secret data
            if bitIndex <= length(secretBits)
                stegoImage(i, j) = pixelValue +
bin2dec(num2str(secretBits(bitIndex)));

```

```

        bitIndex = bitIndex + 1;
    end

    else
        % H: Add secret data
        if bitIndex <= length(secretBits)
            stegoImage(i, j) = pixelValue +
bin2dec(num2str(secretBits(bitIndex)));
            bitIndex = bitIndex + 1;
        end
    end
end
end

% Calculate PSNR and SSIM before compression
psnrValue = psnr(stegoImage, coverImage);
ssimValue = ssim(stegoImage, coverImage);

% Display the PSNR and SSIM values before compression
fprintf('PSNR before compression: %.4f dB\n', psnrValue);
fprintf('SSIM before compression: %.4f\n', ssimValue);

% Compress the stego image using transform coding (DCT)
compressedStegoImage = compressUsingDCT(stegoImage);

% Convert compressed image back to uint8 to match the cover image class
compressedStegoImage = uint8(compressedStegoImage);

% Calculate PSNR and SSIM after compression
psnrValueAfter = psnr(compressedStegoImage, coverImage);
ssimValueAfter = ssim(compressedStegoImage, coverImage);

% Display the PSNR and SSIM values after compression
fprintf('PSNR after compression: %.4f dB\n', psnrValueAfter);
fprintf('SSIM after compression: %.4f\n', ssimValueAfter);
end

% Function to compress image using Transformation Coding
function compressedImage = compressUsingDCT(image)
    % Apply DCT block-wise
    blockSize = 8; % Block size for DCT
    dctImage = blkproc(image, [blockSize blockSize], @dct2); % Apply 2D
    DCT to each block

```

```

% Quantize DCT coefficients
quantizationMatrix = ones(blockSize, blockSize) * 20; % Simplified
quantization matrix
quantizedImage = blkproc(dctImage, [blockSize blockSize], @(block)
round(block ./ quantizationMatrix));

% Dequantize to reconstruct
dequantizedImage = blkproc(quantizedImage, [blockSize blockSize],
@(block) block .* quantizationMatrix);

% Apply inverse DCT block-wise to reconstruct the image
compressedImage = blkproc(dequantizedImage, [blockSize blockSize],
@(idct2); % Inverse 2D DCT
end

```

Figure 9. Compression and Calculating PSNR

```

function embedAndCalculatePSNR_NoFuzzy(coverImagePath,
secretDataPath, stegoImagePath)
% Load the cover image
coverImage = imread(coverImagePath);

% Read the secret data from a text file
fileID = fopen(secretDataPath, 'r');
secretDataBits = fscanf(fileID, '%1d');
fclose(fileID);

% Convert secret text to its ASCII values and then to binary
secretData = double(secretDataBits); % Convert to ASCII numeric values
secretBits = dec2bin(secretData, 8) - '0'; % Convert to binary, 8 bits per
character
secretBits = secretBits(:)'; % Flatten to a row vector

% Initialize the stego image
stegoImage = coverImage;

% Initialize the embedded bits counter
embeddedBitsCount = 0;

% Embed the secret data (uniform rule: replace 2 LSBs for all pixels)
bitIndex = 1;
for i = 1:size(coverImage, 1)

```

```

for j = 1:size(coverImage, 2)
    pixelValue = coverImage(i, j);

    % Uniform embedding: Replace 2 LSBs
    if bitIndex + 1 <= length(secretBits)
        stegoImage(i, j) = bitset(pixelValue, 1, secretBits(bitIndex));
        stegoImage(i, j) = bitset(stegoImage(i, j), 2, secretBits(bitIndex +
1));
        bitIndex = bitIndex + 2;
        embeddedBitsCount = embeddedBitsCount + 2; % Increment by 2
    bits
    end
end
end

% Save the stego image
saveStegoImage(stegoImage, stegoImagePath);

% Calculate PSNR, MSE, and SSIM
psnrValue = psnr(stegoImage, coverImage);
mseValue = immse(stegoImage, coverImage);
ssimValue = ssim(stegoImage, coverImage);

% Calculate embedding capacity (in bits and bits per pixel)
totalPixels = numel(coverImage);
embeddingCapacityBits = embeddedBitsCount;
embeddingCapacityBpp = embeddingCapacityBits / totalPixels;

% Display the results
fprintf('PSNR: %.4f dB\n', psnrValue);
fprintf('MSE: %.4f\n', mseValue);
fprintf('SSIM: %.4f\n', ssimValue);
fprintf('Total embedded bits: %d bits\n', embeddingCapacityBits);
fprintf('Embedding capacity: %.4f bits per pixel (bpp)\n',
embeddingCapacityBpp);
end

function saveStegoImage(stegoImage, outputPath)
    % Ensure the output file path has a .tiff extension
    [~, ~, ext] = fileparts(outputFilePath);
    if ~strcmp(ext, '.tiff') && ~strcmp(ext, '.tif')
        outputPath = strcat(outputFilePath, '.tiff');
    end
end

```

```
% Save the stego image in TIFF format
imwrite(stegoImage, outputPath, 'tiff');
fprintf('Stego image saved as %s\n', outputPath);
end
```

Figure 10. No Fuzzy Logic and Calculating Evaluation Metrics

AUTHOR BIOGRAPHY

Name : Mardhatillah Shevy Ananti
Place, Date of Birth : Kraksaan, 29 March 2003
Gender : Female
Telephone : +6282121154760
Email : mardha.ananti@gmail.com

ACADEMIC

Department : Department of Informatics
FTEIC , ITS
Batch : 2021
Semester : 8 (Eight)